Neural models for information retrieval : towards asymmetry sensitive approaches based on attention models

Thiziri Belkacem

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *28/11/2019* **par :**

Thiziri BELKACEM

Neural Models for Information Retrieval: Towards Asymmetry Sensitive Approaches Based on Attention Models

---

JURY

Dr. HDR. Anne-Laure Ligozat	Maîtresse de conférences HDR d'Université Paris-Saclay	Rapporteure
Pr. Eric Gaussier	Professeur d'Université Grenoble Alps	Rapporteur
Pr. Gaël Dias	Professeur d'Université de Caen Normandie	Examinateur
Pr. Mohand Boughanem	Professeur d'Université Paul Sabatier	Directeur de Thèse
Dr. Taoufiq Dkaki	Professeur Associé d'Université Paul Sabatier	co-Directeur de Thèse
Dr. Jose G. Moreno	Professeur Associé d'Université Paul Sabatier	Encadrant

---

École doctorale et spécialité :
MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture
Unité de Recherche :
Institut de Recherche en Informatique de Toulouse (UMR 5505)
Directeur(s) de Thèse :
Pr. Mohand Boughanem, Dr. Taoufiq Dkaki
Rapporteurs :
Pr. Eric Gaussier et *Dr. HDR. Anne-Laure Ligozat*

ACKNOWLEDGEMENT

Firstly, I would like to express my sincere gratitude to my thesis supervisors Pr. Mohand Boughanem, Dr. Jose G. Moreno, and Dr. Taoufiq Dkaki, for the continuous support they gave me since the Master's degree as well as during the last three years preparing this thesis. I would like to thank each one of them for the continued assistance of my Ph.D. study and related research, for their patience, for all the advice and motivation, and immense knowledge that they shared with me. Their guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisers and mentors for my Ph.D study.

Besides my advisers, I would like to thank the rest of my thesis committee: I thank the referees Dr. HDR. Anne-Laure Ligozat and Pr. Eric Gaussier for agreeing to review our research works, and the examiner Pr. Gaël Dias to evaluate our work. I thank them all for the insightful comments and encouragement, but also for the hard question which incited me to widen my research from various perspectives.

My sincere thanks also goes to Dr. Gilles Hubert the IRIS team leader and all the other permanent members: Pr. Lynda Tamine-Lechani, Dr. HDR. Karen Pinel-Sauvagnat, Dr. HDR. Guillaum Cabanac, and Dr. Yoan Pytarch, who welcomed me as an intern during the preparation of my Master's project, who gave me access to the laboratory and research facilities, and who continued to support me during the preparation of this thesis. I thank them all for their good relationships, team spirit and warm exchanges that we have had during all these four years of work within the same team.

I thank my fellow lab-mates in for the stimulating discussions, for the restless periods we were working together before deadlines, and for all the fun we have had while working all together in the last four years.

Last but not the least, I would like to thank my family: my dear parents without whom I would never have arrived here, those who gave me life but who also made sure to make it beautiful and impressive, my father my source of life knowledge and experience, of positive energy and insight; and my mother my source of tenderness, patience, smiling and joy. I also thank all my brothers and sisters, who are my dear and precious friends, for supporting me spiritually throughout the adventure that I have preparing this thesis and my life in general. And I will not fail to thank someone who encouraged me a lot and supported me too, thank you very much Smail for all the good moments shared together despite my busy schedule and my thesis stress.

KB: Ulama berriket tzitwit, tamment-is zidet.
FR: “Meme si l’abeille est brune (noire), son miel est bon (sucré)”
EN: “Even if the bee is brown (black), its honey is good (sweet)”
KB: Ulac win izegren asif ur yellixs.
FR: “Personne ne peut traverser une rivière sans se mouiller”
EN: “No one can cross a river without getting wet”
(Kabyle sayings)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I

Preface

INTRODUCTION

Thesis Context

In this thesis, we are interested in problems related to text representation and matching, and several related applications in information retrieval, such as question answering and document retrieval.

Information retrieval (IR) is a field of computer science that allows an IR system (IRS) to select, from a collection of documents those that likely correspond to a user's information need [6]. More specifically, IR is mainly the activity of obtaining information resources, such as documents, passages and answers, that are supposed to match an information need expressed as a query or a question. IR models are proposed and implemented to cope with the main purpose of IR, retrieving relevant information w.r.t. a query. Most of IR models are based on the same assumption, documents and queries are represented as weighted bag of words (BoW) and the relevance is interpreted as same as lexical matching between query words and document words. These models estimate the relevance score based on the exact matching between words of the input text sequences. Hence, some returned documents by the IRS do not always address the search intent of the user's query. This limitation is due the BoW representations that consider every text sequence as a set of independent words. The main problem resulting from these approaches is the *vocabulary mismatch*. It consists in using a different vocabulary in the query (or question) and its corresponding relevant results (documents or answers).

To overcome the BoW and *exact* matching limitations, semantic based approaches using external resources such as WordNet have been proposed [7]. These approaches are resource consuming and lack of genericity. Recently, new approaches based on machine learning algorithms have been used in order to learn vector representations, namely *embedding* models [2, 8, 9] that leverage the semantic feature of different words in a text, allowing semantic links between text sequences to be expressed via vector operations. In addition, advances in image processing, through neural networks and deep learning [10], have inspired research in the different fields of text matching [11] to explore several *deep* models to take advantage of the computational ability and the non-linearity of these models, in order to capture different latent characteristics related to text representation and matching.

Research Issues

In this thesis, we address three main issues related to different text matching tasks: (1) the impact of using embedded representations of words in IR models; (2) the impact of the neural model's architecture w.r.t. different matching tasks; (3) dealing with the document length wile using neural models for ad-hoc document ranking. In the following, we describe each of these main issues in more details.

1. Embedded Representations in Text Matching

Recent text matching models largely use the distributed representations of words [2, 9] and sequences [8, 12]. These embedded representations, enable the matching models to consider the semantic relatedness between two different texts since similarities between words are computed based on their vector representations. In the context of document retrieval, the proposed models [13, 14, 15] represent documents and queries by their component word vectors, and their similarities are computed based on vector computations that handle all the word vectors of the query and the document in the same way. This may solve problems related to exact matching. However, several questions may rise on how combining both query word vectors and document word vectors when computing their similarity. Indeed, we believe that there is a key difference between the query words and the document words. In fact, query words that do not appear in the document may have a different impact and contribute differently to the document content, compared to those that are present in this document.

In this thesis, we first analyze the different ways that the distributed representations of words and sequences have been used, in text matching. Then, we propose a different matching strategies, combining the semantic-based word matching with the exact matching, in order to analyze the contribution and impact of query terms in the matching function.

2. Neural Models and Text Matching Tasks

Deep learning is gaining a large interest in recent NLP and IR. Several models based on different architectures have been proposed to handle a bunch of tasks related to text matching. We distinguish mainly the simple feed-forward models [16], the convolutional models [17] and the recurrent models [18]. In most of these models, the input sequences undergo the same transformations to construct the corresponding representations, independently from the task being addressed. The nature of the task can be defined w.r.t. the relationship between the input text sequences and their types. Specifically, we distinguish two types of matching: (1) the *symmetric* matching refers to matching tasks where inputs are of the same nature, such as paraphrase identification and document classification ; (2) the *asymmetric* matching refers to tasks where inputs are of different natures, such as ad-hoc document ranking and answer sequence selection. Most neural text matching models in the state-of-the-art [3, 19, 20, 21, 22] have adopted a Siamese architecture [23] due to its simplicity. In these models, both input representations are constructed in the same way, from the vectors of their component words. The models do not consider the differences between text matching tasks in general. The complementary relationship between some

text sequences, such as the query and the document or the question and the answer, is not taken into account.

In this thesis, we first describe the *asymmetry* aspect related to some text matching applications, and then propose an approach to take into account the type of the matching task, (1) or (2), to better process the inputs and perform the matching.

More recently, position-based models [3, 24] and attention-based models [25, 26], have been proposed, taking into account the word position in a text, in the former, and the importance of a word in the text sequences, in the latter. Position information of a word in a text, is one of important features that help text representation learning. However, the importance of a given piece of information among others in a text is not considered in position features, as it is made in the attention models. In these latter, the attention weights help to identify the core information to be considered in a given text sequence, and thus can be used with position features in order to help the model to focus in the most important information while processing the different positions of a sequence. However, none of the proposed neural models for text matching have combined the position information with the attention weights.

In this thesis, we combine the position-based representation learning approach with the attention-based model. We believe that when the model is aware of both word position and importance, the learned representations will get more relevant features for the matching process.

3. Dealing with Document Lengths in Neural Models

Most neural IR models struggle to deal with the variation of documents length. Long documents may cover several topics which over complicate the matching task and decrease performances of current neural IR models. To overcome this problem, the common solution [21] is to cut off the exceed text in order to make the document content shorter and hence the model more effective.

Previous analysis [21, 27] have shown that long documents usually contain relevant parts far from their beginning, and suggest to assess the document relevance at different information granularities, in particular, the word-level and the passage-level. Several passage retrieval models [28, 29, 30] have been proposed in order to deal with the document length problem in IR, but these models are not able to capture the different semantic links along the document, such as the relatedness between different passages of a document and their relevance features. These models are based on simple aggregation functions to combine the passage-level relevance scores.

In our work, we propose a matching approach assess the document relevance at different levels w.r.t the information granularity in a document and address the document length problem.

Main Contributions

The main contribution of this thesis is the definition of text matching models using neural networks and distributed representations of texts. We proceed in three main stages: (1) analyzing the use of distributed word representations in classical IR models, and pointing out how these representations contribute to

the performance of different models; (2) studying the matching of short texts using neural networks, and how the different matching tasks are handled by existing models; (3) proposing a neural model that better handle document length variation. At each of these stages, a solution is proposed to solve the problems encountered by existing methods. More precisely, our contributions are as follows:

1. *Word embedding in classical IR models.* First, we analyze the importance of query terms that are not present in the relevant documents while performing the document-query matching, using the distributed word representations. To do this, we use different document-query matching strategies. The objective is to treat the terms of the query that are absent in a document differently from those that are present in it. We combine the *exact* matching of classical IR models with the *semantic* matching of distributed word representations. The matching strategies are evaluated and compared to state-of-the-art models, using different TREC datasets. We specifically use news datasets, such as Robust and AP, as well as the GOV2 web dataset. Results support our hypothesis about the differences between query words.

2. *Neural models for text matching.* In a second step, we analyze the neural text matching state-of-the-art and identify the main differences between text matching applications. Using the attention-based models [31], we made two main contributions:

   - *Asymmetry Sensitive Architecture for Neural Text Matching.* We first describe the *asymmetry* aspect of several text matching tasks. More specifically, *asymmetric* tasks include document-query matching as well as question-answer matching, and involve a kind of *complementary* relationship between the input sequences that the model can leverage. *Symmetric* tasks, include document classification and paraphrase identification, and aim at identifying if both input text sequences mean the same thing or refer to the same information. To handle the differences of these matching tasks, we propose an *asymmetry* sensitive architecture enabling to extend different neural text matching models using attention layers. We further extend several well known models, and show the interest of taking into account the nature of the task being addressed, in the matching model.

   - *Attention-based matching using Multiple positional Vector representations.* In a second step, we analyzed the impact of combining position features and attention weights of different words in text sequences, on the matching tasks. Extending the position-based model MV-LSTM [3] with attention layers, within the *asymmetry* sensitive architecture described above, we proposed made a comparative analysis of the MV-LSTM model with and without the attention features.

   Experimental results, in question-answer matching and question pairs identification tasks, show that our asymmetry sensitive architecture enables to enhance the performances of different extended models using the attention layers. Besides, while combining the position features with the

attention weights, the model performs better on ranking relevant information.

3. *Multi-level Relevance Assessment of Long Documents.* Finally, in order to cope with the problem of document length in neural text matching models, we focus on two main aspects, and proceed accordingly: (a) identifying the most important elements in a sequence using attention models; (b) the fact that a long document can deal with several topics and thus can contain different relevant passages. We proposed two main approaches:

   - *A Multi-level Attention-based Architecture.* We consider passage-based document representations, where every document is viewed as a set of its passages that are likely to be relevant, as in [21, 27]. We propose an attention-based architecture to extend several neural text matching state-of-the-art models, in order to perform a multi-level relevance assessment. To do this, we use attention layers at the word-level as well as the passage-level. The attention weights enable us to identify the most important words of every input sequence (query or passage), in the former, and passages to be considered in the matching layer, in the latter.
   - *An Attention-based Multi-level Matching Model (AM3).* Then, we propose AM3, a multi-level neural matching model that exploits three different levels of information granularity: the *word-level*, the *passage-level* and the *document-level*. Considering a long document as a set of passages, our AM3 model focuses on the most important words of the query and the passage, using attention weights. At the word-level, we compute attention weights for the different words in the query and the document. At the passage-level, we first compute a matching matrix between all words of the query and the different document passages. Then, we represent every passage by an embedded vector computed using a set of filters applied to the matching matrix, and we compute attention weights for the different passages to highlight the most relevant ones. Finally, at the document-level, we use an interaction layer in order to compute the final matching score of the whole document.

   We evaluate these contributions using a news-wire TREC dataset (Robust), and the experimental results show the advantage of using the passage-based document representation, compared to the use of the document content set to a maximum size. Furthermore, the extended models, within the proposed architecture outperform the corresponding original counterparts.

Thesis Overview and Organization

This document is made of three main parts and a conclusion. The *Background* part II contains two chapters describing the basic concepts in IR and neural networks; the *State of The Art Overview* part III contains two other chapters describing the text representation models and the deep models in text matching; the *Contributions* part IV contains a first chapter describing all the experimental framework that is used in this work, and three other chapters describing

our contributions and analysis. Finally, the *Conclusion* part V gives a general assessment of the work carried out during the preparation of this thesis, and how the proposed models contribute to the resolution of the various issues that we addressed. In the followings, we describe each of the different chapters separately:

- Chapter 1 describes several basic concepts in information retrieval. First, we provide a set of basic definitions (section 1.2), then describe the main text representation methods (section 1.3), and the matching process (section 1.4). Section 1.5 describes the main objective of evaluating an IR system, and a set of evaluation measures and campaigns. Different issues related to text matching are discussed in section 1.6.

- Chapter 2 describes a set of basic concepts about artificial neural networks (NN) and deep learning. We give a set definitions related to these models in section 2.2. In section 2.3, we describe a set of most used neural architectures, and we focus mainly on those used in this thesis. In section 2.4, we describe the NN training process and methods, and give a set of widely used algorithms in section 2.5. In section 2.6, we explain the over-fitting problem of neural models and describe some methods used to solve this problem.

- Chapter 3 concerns text representation approaches. It presents a set of models that are used to learn distributed representations of text, at the word-level (section 3.2) and the sequence level (section 3.3). In section 3.4 we discuss how these representations have been used in different text matching applications. We provide a taxonomy describing several state-of-the-art models that use these representations. Issues related to these representations are listed in section 3.5 and we end with a discussion in section 3.6.

- Chapter 4 gives first a description of the learning to rank (LTR) framework in section 4.2. In section 4.3 we give an overall overview of the deep learning applications in text, in particular for representation learning and matching, and define a unified deep model for text matching, that supports different state-of-the-art models. In section 4.3, we give a taxonomy of deep models for text matching based in two main criteria, the architecture and the representation features. In section 4.4, we discuss several issues related to deep models in text matching.

- Chapter 5 describes the experimental setup and frameworks used in the different experiments. In section 5.2, we describe the different datasets. In section 5.3 we give a list of evaluation metrics that we use to evaluate performance of the different models. In section 5.4, we give an overall description of the different baseline models, specifically those of the classical models, the classical models using word embeddings and the neural models. Finally, in section 5.5, we describe the list of frameworks and technical tools.

- Chapter 6 we perform an empirical analysis of the impact of word embeddings in classical IR models. We based our analysis on the different

contributions of query words that are present in relevant documents compared to those that are not. We define three different matching strategies using the semantic similarities between embedded word vectors, in section 6.4. Finally, in section 6.5, we describe the experimental results and analysis, where we evaluate the impact of different parameters used in each of the matching strategies we proposed.

- Chapter 7 explains our main contributions about neural models for short text matching. First, in section 7.2 we propose a matching approach that enables several neural matching models to handle the differences between text matching tasks. We define the *asymmetry* aspect of several text matching tasks, in section 7.2.1. In section 7.2.2, we describe our approach to better address the different matching tasks, and extend several neural matching models. We describe the training process of the different models in section 7.3. In section 7.4, we analyze results of the different models separately, and perform an empirical analysis of an attention-based model with multiple positional representations, for matching sequences.

- Chapter 8 describes a matching model for ad-hoc ranking of long documents. In section 8.2, we describe our multi-level matching architecture enabling us to extend several text matching models with attention layers, in order to perform a multi-level relevance assessment. Furthermore in section 8.3, we propose AM3 an attention-based matching model, where word-level, passage-level, and document-level signals contribute to the final matching score. Finally, in section 8.4, we describe the experimental part and give a detailed analysis of the obtained results.

- Conclusion and future work are listed in part V.

Part II

Background

CHAPTER 1

BASIC CONCEPTS IN INFORMATION RETRIEVAL

1.1 Introduction

Information retrieval (IR) is defined as *finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections of documents (usually stored on computers)* [32].

First approaches in IR were proposed for text search [33]. The main objective of such a task is to find relevant results that satisfy an information need expressed in a *query*. In order to perform the retrieval task, an IR system (IRS) performs these main steps: once the dataset is indexed, a query and document representations are constructed. Then perform the query-document matching, and finally rank the candidate documents. This process is highlighted in figure 1.1. In this figure, the indexing step creates a token-to-documents list. This index can help finding a document terms. Representations are built first, for both the query and the documents. These representations are then matched and a score is assigned to every document-query pair. These scores are supposed to reveal the relevance of a document w.r.t. the query. Finally, the documents are ranked according to the relevance score and the top k results are returned to the user. If the user is not satisfied with the results, the query may be reformulated [34], based on the user's feedback [35] or an external resources [36]. Then a new search iteration is performed in order to return new results that could more satisfy to the user.

In the following, we consider a set of notations defined in table 1.1. We will first present several basic concepts related to the IR process. Mainly the relevance concept, the IR matching models and evaluation, then we end up with some issues that an IRS needs to handle.

1.2 Definitions

1.2.1 Sequence

We refer with *sequence* to the different granularities of the text representation, including sentences, passages, paragraphs and documents. Thus, we consider as

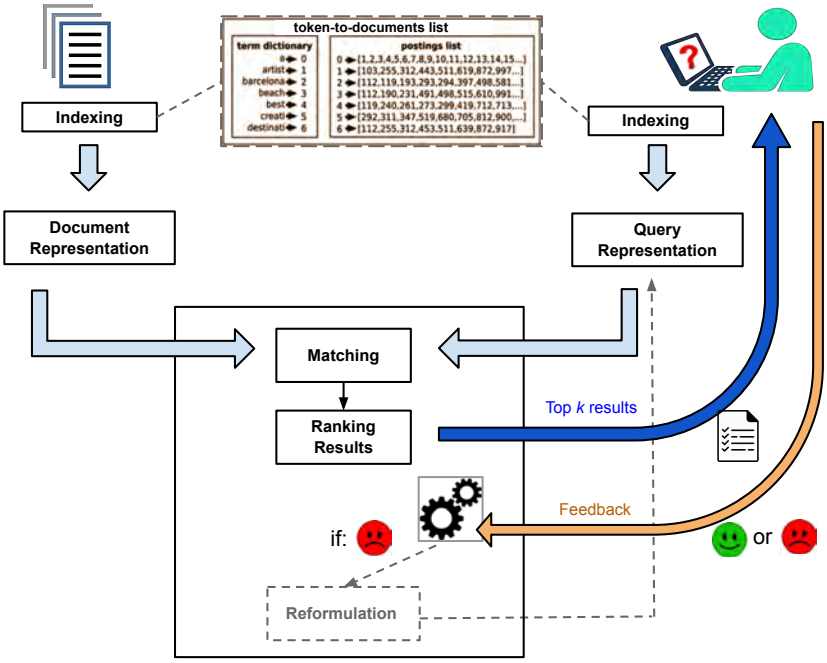


Figure 1.1: Description of the basic process of information retrieval systems.

Notation	Description
$\|S\|$	Cardinal of the set S
w_i	the ith word of a sequence
$s =< w_1 ... w_{\|s\|} >$	sequence of words
$\{w_1, ..., w_n\}$	set of words
Q	query
D	document
q_i	a query word
d_j	a document word
V	set of all vocabulary words
RSV	relevance status value

Table 1.1: Set of notations.

a sequence any list of words that follow each other in a precise order, in order to express a given information.

1.2.2 Document

A document is defined in *Wikipedia*[1] as *"everything that may be represented or memorialized in order to serve as evidence".* Several literal definitions have been provided [37, 38]. Documents can be digital or paper. In [39], Levy have provided the main characteristics of a digital document, compared to paper one. In IR, documents are the central elements of any IRS. In this work, we consider digital document, such that every document is considered as sequence of words $D =< d_1...d_{|D|} >$.

1.2.3 Query

A query is a sequence of words that are used by the user in order to express his/her information need. Some studies [40, 41] have shown that the most common length of a query is about two words [40], and multiple queries could be used in order to express the same information sought [41].
As for the document, the user query is also a sequence of words $Q =< q_1...q_{|Q|} >$ with $|Q|$ is much smaller than $|D|$.

1.2.4 Relevance

The concept of *relevance* is central to IR. It is defined as a "*correspondence in context between an information requirement statement and an item or element, i.e. the extent to which the item covers material that is appropriate to the requirement statement*" [42]. Cooper [43] have given several definitions of *relevance* from different resources.
We can distinguish two main definitions of relevance [44]: 1) the *system relevance* (relevance as seen by the IRS) corresponding to the scoring value computed for a given document w.r.t. a given query. 2) the *user relevance* (relevance according to the user) referring to the satisfaction of his/her information need expressed in the query. The analysis of Mao et al. [45] highlights the differences between the system relevance of a result and the user satisfaction.

1.3 Text Representation

In order to perform the query-document matching, their contents (texts in our case) are first mapped to internal representations. Several text representation models have been used in IR models. In this section, we present two main classes among others: the first one is the classical Bag-of-Words representation used in several exact matching-based models [46]. The second class is about distributed representations for semantic-based matching [47].

[1] https://en.wikipedia.org/wiki/Document#Abstract_definitions

1.3.1 Bag-of-Words (BoW) Representations

BoW is the common approach used for representing text in most classical text matching tasks, in general and information retrieval in particular. A BoW is a structurally simple representation constructed without linguistic or domain knowledge skills. It consists in representing every input sequence s using a set of independent words $\{w_1, ..., w_n\}$. Where w_i is a given word of a sequence s as defined in table 1.1. Hence, every word sequence can be represented based on *one-hot* word vectors whose dimension are equal to the size of the dataset vocabulary. The only non null value of every vector is either 1 or a real value representing the weight or frequency of each word in the dataset. This kind of representations have been used in several text matching models including text classification [48], document ranking [46, 49] and question answering [50]. While using the BoW representations, the texts similarity is based on exact matching of words of the input sequences, and that consists on leveraging only lexical similarities. This matching approach does not take into account the semantic relatedness of words and their meanings. This leads to a well known *vocabulary mismatch* problem. In IR, this problem happens when words that are used in the query are different from those used in relevant documents, and where synonyms of the query words can be used instead. Besides, some words of the query can be polysemous and hence, be used in documents that are not relevant to the query. The exact matching does not take into account these aspects, and a document having many or all query words is expected to be relevant. Nevertheless, some documents could contain query words without necessarily being relevant for that query, while other documents could contain no query word and be relevant. To cope with these issues, semantic-based matching approaches have been proposed taking into account synonyms and words of the same context.

1.3.2 Semantic-based Representations

Semantic-based similarities between words could be used to cope with the vocabulary mismatch problem. Several models have been proposed to construct word representations that are capable to better handle the semantic similarities between words and sequences.

The commonly called *word embedding* [51] or *distributed word representations*, are a vector representation of words used to capture fine-grained semantic and syntactic regularities. These models construct a semantic-based vector space where every word is represented with a real-valued vector. There are two main approaches for building distributed word representations: 1) the global matrix factorization approach, such as Latent Semantic Analysis (LSA) [47] and GloVe [9], 2) the local context window approach, such as Word2Vec [52].

LSA [53] word vector representations are based on a singular value decomposition (SVD) method applied to a document-word matrix. The LSA-based approaches assume that the vectors dimensionality is highly valuable. The main idea behind this assumption is that reducing the dimensionality of the observed data from the number of initial contexts to a much smaller number will produce better approximations of semantic similarities [54]. GloVe [9] is a global log-bilinear regression model that combines different aspects about the global matrix factorization methods and the local context window methods. It leverages statistical information by training the model only on the nonzero elements

in the global word-word co-occurrence matrix, rather than on the entire sparse matrix.

Differently from matrix factorization-based methods, models based on the local context window method, such as the Word2Vec model [55] and the Log-BiLinear model [56], represent words according to their contexts (neighbors), and learn word vector representations through a neural network architecture. These models have demonstrated the capacity to learn linguistic patterns as linear relationships between the word vectors [52].

1.4 Text Matching Process

Text matching aims to compare two text sequences based on several criteria. It is the core of many tasks. A large set of text matching models are proposed in the literature. We present a generic model to explain the text matching process. Given two text sequences s_1 and s_2, a representation function φ and a vocabulary space V. Each sequence is represented by a vector $\varphi(s_i) \in \mathbb{R}^{|V|}$, if a BoW representation scheme is used; or a matrix $\varphi(s_i) \in \mathbb{R}^{V \times dim}$, if a distributed representation scheme is used.

The similarity between the two sequences, named relevance status value (RSV), is then computed as described in function 1.1. This assessment takes into account the importance weights of every term of s_1 in s_2.

$$RSV(s_1, s_2) = M(\varphi(s_1), \varphi(s_2)) \tag{1.1}$$

In case of document retrieval task, documents of the dataset are ranked based on their relevance scores $RSV(Q, D)$ and top k ones are returned to the user by the IRS.

Several models have been proposed in text matching literature to compute the RSV value for matched sequences. The classical matching models [57, 49] are refereed to as *tf.idf* model, since they rely on word statistics such as the *term frequency* (tf) and the *inverse document frequency* (idf). The *idf* weighting, firstly introduced by Sparck Jones in 1972 [46], is based on empirical observations of the global term frequency *tf*, whereby highly frequent terms in a collection should be given less weight than less frequent terms that are more common and less discriminative. The work that followed in IR [57, 49, 58] have highly used this perception. Some of the well known *tf.idf* models are the vector space model [59], the probabilistic BM25 model [57], and the language modeling LM [60]. Recently, several models based on machine learning techniques have been proposed and used in several text matching applications [61, 62, 63].

We are interested in classical matching models using distributed representations as well as neural models. A detailed description of these models is provided in the chapter 4.

1.5 Evaluation in IR

The quality of the documents ranking performed by a matching model is essential. Indeed, the user usually considers only the first page of ranked documents (the top 10 or 20) by a web search engine [64]. If relevant documents are not present in the first page, the user will not be satisfied by the results returned

by the system. Several measures and practical benchmarks have been proposed in order to evaluate how effective is an IR approach.

1.5.1 Evaluation Measures

Several evaluation measures are used in order to assess the effectiveness of an IRS. In this section, we present some of widely used measures in IR. We group them according to their use, *results-based* or *ranking-based*:

Results-based

Results-based evaluation measures evaluate the overall set of returned documents per query. The objective is to measure how an IRS is capable to find all relevant documents and reject all irrelevant documents.

- *Recall & Precision.* [65] Given ρ a set of retrieved documents by a given system, *precision* P is the fraction of relevant documents that have been retrieved, ρ^+, over the total amount of returned documents, while the *recall* R corresponds to the fraction of relevant documents that have been retrieved among all relevant documents.

$$P = \frac{\rho^+}{\rho^+ + \rho^-} \tag{1.2}$$

$$R = \frac{\rho^+}{\rho^+ + \bar{\rho}^+} \tag{1.3}$$

  where $\rho = \rho^+ + \rho^-$ is the total number of retrieved documents. ρ^- and $\bar{\rho}^+$, refer respectively to, a number of irrelevant retrieved documents, and a number of relevant documents that were not found for the given system.

- *Acc.* The *Accuracy* is one of the metrics used to evaluate binary classifiers. While in ranking tasks, the objective is to evaluate the ordering of the relevant elements in a list of results, in classification tasks, the evaluation objective is to assess the systems' ability to correctly categorize a set of instances.

  For example, as a binary classification application in text matching, we would like the system to predict the correct label (*ex:* 0 or 1) that reflects an element's relevance. Hence, given a dataset having S positive elements and N negative elements. The *accuracy* (Acc) of a model M could be defined as in equation 1.4.

$$Acc(M) = \frac{TS + TN}{S + N} \tag{1.4}$$

  where, TS and TN are respectively the number of elements that are correctly classified as *positive* ones, and the number of elements that are correctly classified as *negative* ones. Hence, for a total population (evaluation dataset), the closer to 1 is the model's accuracy, the better it is.

The *recall*, *precision* and *Acc* assume that the relevance of each document could be judged in isolation, independently from other documents [66]. However, in [67], Goffman recognized that the relevance of a given document must

be determined with respect to the documents that are returned before that document. In another work [32], Manning et al. argued that the user cares more about good results that are on the first page or the first three pages. This leads to measuring precision at fixed low levels of retrieved results, such as 10 or 30 documents. Hence, the effectiveness must be assessed in different ranks of the returned results list.

Ranking-based

The ranking-based evaluation measures highlight the results ranks and promote relevant models that return documents at the top of results list.

- *P@k.* The objective is to compute, at a given rank position k, the corresponding precision value P. It is computed as described in equation 1.2, where only the top k results need to be examined to determine if they are relevant or not [32].

- *MAP.* The mean average precision (MAP) is one of the evaluation measures in widespread use. The MAP computed for a set of n queries corresponds to the mean of the average *precision* scores for each query in this set.

$$MAP = \frac{\sum_{i=1}^{n} AP(Q_i)}{n} \quad (1.5) \qquad AP(Q_i) = \frac{\sum_{k=1}^{n_i} (P@k(Q_i).\varepsilon_k)}{\rho} \quad (1.6)$$

  where n_i refers to the number of retrieved results for the query Q_i. $P@k(Q_i)$ refers to the precision at cut-off k in the corresponding results' list. ε_k is an indicator function which is equal to 1 if the item at rank k is a relevant document, 0 otherwise [68]. ρ refers to the number of total relevant documents.

- *MRR.* The Reciprocal Rank (RR) computes the reciprocal of the rank at which the first relevant document is retrieved for a given query Q_i. RR is equal to 1 if a relevant document is retrieved at the 1st rank, or $\frac{1}{2}$ if a relevant document is retrieved at 2nd rank and so on. When averaged across all evaluation queries, the measure is called the Mean Reciprocal Rank (MRR).

$$MRR = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{rank_i} \quad (1.7)$$

  where $rank_i$ refers to the rank position of the first relevant document for the query Q_i.

- *DCG* the Discounted Cumulative Gain (DCG) aims to measure relevant documents ranked highly in the rank list, and penalize highly relevant documents appearing lower in a search result list by means of the graded relevance value which is reduced logarithmically proportional to the position k of the result [69].

$$DCG@\text{k} = \sum_{i=1}^{k} \frac{\varepsilon_i}{\log_2(i+1)} \quad (1.8)$$

where ε_i refers to the relevance value of the result at position i.
Since search result lists vary in length depending on the query, the normalized DCG (nDCG) computes the cumulative gain that the user obtains by examining the retrieval result up to a given ranked position k [69] .

$$nDCG@\text{k} = \frac{DCG_k}{IDCG_k} \tag{1.9}$$

where $IDCG@k = \sum_{i=1}^{|REL_k|} \frac{2^{\varepsilon_i}-1}{\log_2(i+1)}$ is the ideal discounted cumulative gain, where REL_k represents the list of relevant documents in the corpus up to position k, ranked in a descending order of their relevance.

1.5.2 Benchmarks and Campaigns

The Text Retrieval Conference (TREC)[2] provides a complete evaluation benchmark to compare the different proposed IR methods. The TREC datasets result from a project initiated by the Defense Advanced Research Project Agency (DARPA)[3], co-organized by the National Institute of Standards and Technology (NIST)[4]. The NII Testbeds and Community for Information access Research (NTCIR)[5] is a series of evaluation workshops designed to enhance research in information technology including mainly information retrieval and question answering. A grid of several evaluation campaigns launch, for text processing research, could be found in [70].

1.6 Text Matching Issues

We can distinguish several limitations related to classical text matching methods. First, the widely used BoW representation approach considers the words of a text as independent elements. However, words of a given text are some how related to each other. We believe that characteristics of the different words in a sequence, such as the sequencing, positions and co-occurrences are some of the most important aspects that could help to distinguish the relevant document from other documents that contain the query words without necessarily being relevant. Besides, the BoW-based matching models [57, 50, 48] are statistical methods where the query words frequency in a document is the basic asset to compute its relevance. These models are based on the exact matching and consider documents with no query word as irrelevant. However, several documents could be relevant without having any lexical match information with the query. For instance, synonyms of the query words can be used instead of these words themselves. Hence, a semantic-based matching scheme is needed in order to cope with these limitations.
To overcome the aforementioned problems, semantic-based methods, such as latent semantics LSI [71] and LSA [53], concept-based methods [7, 72], and word-sense matching [73], can provide a way to represent a text as a set of concepts, where word representations are computed based on a word co-occurrence matrix factorization, or external resource such as thesaurus. These models enable to

[2] https://trec.nist.gov/
[3] https://www.darpa.mil/
[4] https://www.nist.gov/
[5] http://research.nii.ac.jp/ntcir/index-en.html

efficiently leverage statistical information in a text corpus but perform relatively poorly in word analogy tasks [9]. Recently, context-based representation models [52, 56] have been proposed. These models enable to learn semantic-based word representations allowing to effectively compute the semantic relatedness between lexically different words and reveal the contextual similarities and links. Hence, these representation models can be used to overcome limitations of the classical BoW and latent semantic models.

Beyond the word-based similarity, IR approaches are confronted with a variety of issues related to the text matching process. First, the gap between the query length and the document length makes it difficult to retrieve the appropriate document for the user query. More specifically, the query is shown [74] to be regularly short and made of a small amount of key words, which leads sometimes to ambiguity problems. In contrast, the document is much longer and could span different topics. Hence, these documents may be entirely or partially relevant. Such that, they may have passages that are likely more relevant to the query than other passages of the same document. Besides, in the same document or word sequence in general, words are not of the same importance, and we need to handle them accordingly in a matching model. Finally, most text matching models handle the sequences being matched in the same way regardless of the nature of the matching task. In chapter 7 we highlight the differences between several matching tasks and the related issue.

To handle the different aspects aforementioned, text representation and matching models need to process input sequences deeper than the simple word-level analysis and the lexical similarity assessment. To do so, recent text matching models tend to use automatic and deep learning techniques, to build models for text representation learning and relevance assessment of different results (documents, answers). In chapter 4, we give an overall description of these models and discuss several related issues.

1.7 Conclusion

In this chapter, we have viewed the basic concepts in IR, from the input representation to the results retrieval. We have seen the main limitations of the classical BoW representation approach and the exact matching process. In fact, the simplicity of these classical methods does not allow the matching models to capture semantic relatedness as well as the contextual information that are important and could influence the retrieval process. To overcome these limitations, approaches based on word co-occurrences represent a document as a set of concepts. The objective of these approaches is to explore the semantic relatedness between words while performing the text matching process.

The recent IR research interest is focused on using machine learning and deep learning techniques in order to solve several issues related to classical text matching models. Several deep models have been proposed exploiting both the semantic representations and the neural networks. The objective of the neural-based models is to capture higher level semantic features and relevance signals, for better performing the matching function. In the next chapter, we will present a set of basic concepts about neural models and deep learning aspects.

CHAPTER 2

BASIC CONCEPTS IN NEURAL NETWORKS AND DEEP LEARNING

2.1 Introduction

The advances in machine learning have emerged a set of non-linear methods known as *deep structured learning*, or more commonly *deep learning* [75]. These new techniques are based on different structures of artificial neural networks, which are composed of multiple processing layers, to learn representations of data with multiple levels of abstraction [10].

Deep learning methods have affected several artificial intelligence fields such as computer vision [76], speech recognition [77] and machine translation [78]. Artificial neural networks have been introduced in 1943 by McCulloch and Pitts [79]. It is a bio-inspired[1] computational approach, including several models for automatic information processing [80, 81, 51]. Besides, the invention of the Graphic Processing Units (GPUs) [82] is one of the most important factors that encouraged and emerged several developments in the deep learning area. Indeed, GPUs rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device. Their highly parallel structure makes them more efficient than general-purpose central processing units (CPUs), for algorithms that process large blocks of data in parallel such as DNNs.

In this chapter, we aim to present a set of basic concepts of neural models and deep learning. We introduce key definitions and computational aspects. Following this, we present the mostly used neural model architectures. Finally, we present the training process of neural-based methods and some related issues.

[1] Biologically Inspired Computing is a field, related to artificial intelligence, and concerns a study that loosely knits together subfields related to the topics of connectionism, social behaviour and emergence. *Wikipedia*

Notation	Description
x	input vector
x_i	one element (coordinate) of x
y	output vector
y_j	one output element in y
f	transformation function
ψ	combination function
δ	activation function
W	weight matrix connecting two different layers of a network
b	bias vector corresponding to nodes of one layer in the network
h	a hidden state of a given hidden layer in the network
$\pounds$	loss function (objective function)
θ	all the free parameters of a model

Table 2.1: Set of notations.

2.2 Main Concepts and Definitions

2.2.1 Notations

Along this chapter, we adopt a set of notations, most of which are described in table 2.1.

2.2.2 Artificial Neurons

The artificial neuron or a *formal* neuron [83] is a computational unit for non linear data processing. Figure 2.1 shows an abstraction of the functionalities of a biological neuron in a formal neuron.
The set of dendrites in the biological neuron are structures that receive electrical messages through the synaptic connections. These messages correspond to a set of input data values received by the artificial neuron, via every x_i input connection. The cell body on the biological neuron is the element that process the received electrical messages. This process corresponds to the combination and activation process, and which is represented by the function f in the in the artificial neuron of figure 2.1. Finally, the axon terminal of the biological neuron corresponds to the output y of the formal neuron that will be fed to neurons of another layer returned as a final output value.

The neuron model is a mathematical model that receives different input information as a set of digital signals. These inputs are then integrated, using function f, with a set of free and trainable parameters, including the connection weights w_i and the input bias b, to produce a single output. This aspect is highlighted in figure 2.2, where we identify several elements that are involved in transforming the input signals vector $x = [x_1, ...x_n]$ to the single output y.

The Free Parameters

To every neuron, is associated a set of trainable parameters, called *free parameters*, including the connection weight vector w and the bias b which is a connection weight corresponding to an additional (imaginary) input of value 1.

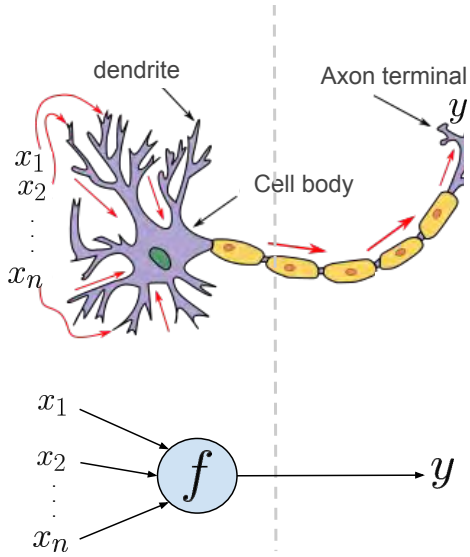


Figure 2.1: Functionality correspondence between a biological neuron and an artificial neuron.

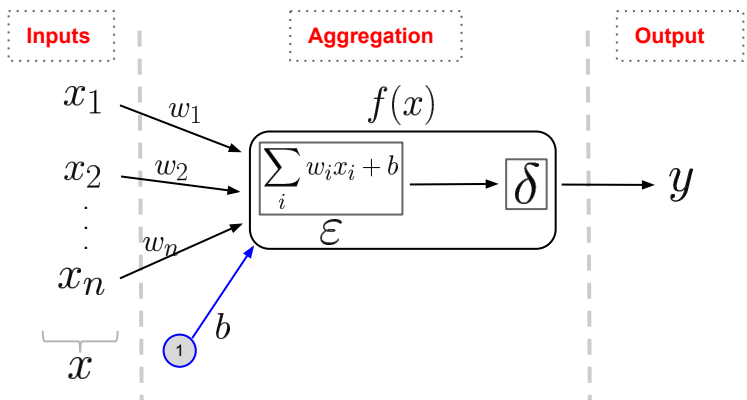


Figure 2.2: A closer look at the operating process of an artificial neuron.

The free parameters are usually referred to as *model parameters*, and denoted as $\theta = \{b, w\} \in \mathbb{R} \times \mathbb{R}^n$.

The Transformation Function

Every artificial neuron includes a function f used to transform the input elements of x to the output y. The function f is in fact composed of two main functions: the *aggregation function* ψ and the *activation function* δ, as shown in figure 2.2. The objective of these functions is to combine the different inputs using a linear function, in the former, and to transform this combination using a non-linear function, in the latter. Hence, f can be defined as in equation 2.1.

$$y = f(x), \quad \text{and} \quad f(x) = \delta(\psi(x_1, ..., x_n)) = \delta\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.1}$$

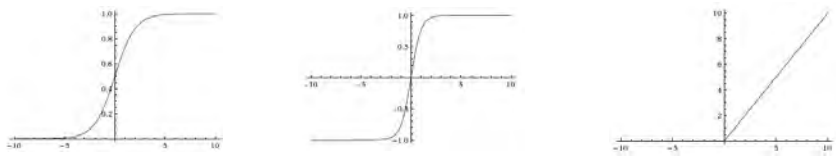
where w_i is the connection weight corresponding to the input x_i.

2.2.3 The Activation Function

The activation function, also called *transfer function*, is used to control the information propagation through a multi-layer network, based on a defined threshold value ϵ. Depending on this value and the output value y_j, the neuron could have several states:

- $y_j < \epsilon$: neuron inhibited or inactive;
- $y_j \approx \epsilon$: transition phase;
- $y_j > \epsilon$: neuron active.

In practice, several activation functions are used depending on the application's objective. Every activation function takes a single number and performs a mathematical operation on it to map it to a corresponding interval. Some of the widely used activation functions are *Tanh*, *ReLU*, and *Sigmoid*. In figure 2.3 we show their corresponding curves.



(a) Sigmoid (b) Tanh (c) ReLU

Figure 2.3: The curves describing the different behaviours corresponding to three different activation functions used in neural networks.

In [84], presents an overview of activation functions and compare them through an empirical analysis using several deep learning applications.

2.2.4 Artificial Neural Networks

Although a single neuron is able of learning and solving some classification problems, it can not model complex links between data and complex tasks though. Neural networks are highly connected graphs where nodes correspond to artificial neurons working in parallel and vertices correspond to the network connections. Each neuron is an elementary processor that computes an output value based on the information it receives in input, as described in figure 2.2. The connections between the different neurons are weighted and make up the network topology, and which can vary from one model to another. Figure 2.4 shows an example of a simple multi-layer perceptron MLP, with three layers. An MLP [85], called also a *simple feed-forward* network [86], consists of neurons that are ordered into layers. Between the input layer and the output layer, there could be several hidden layers all connected in one side only, from current to next layer in the direction of the output.

There are also recurrent architectures where the inputs of a neuron can be its own outputs. These NN architectures are presented in section 2.3 among other widely used models.

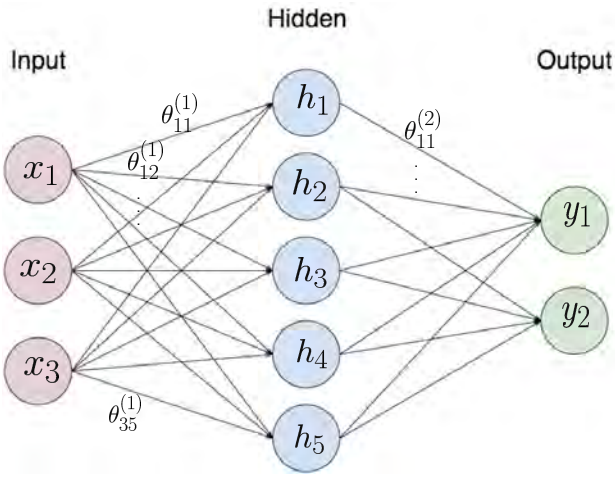


Figure 2.4: Architecture of a multi-layer perceptron MLP with one hidden layer.

In figure 2.4, the *input* layer of the NN corresponds to the input data x, the *hidden* layer contains a set of computed hidden (intermediate) states h_j. The parameters $\theta_{ij} = \{w_{ij}, b_i\}$ include a connection weight w_{ij} and a bias value b_i, corresponding to the node i connected to the node j. The *output* layer gives the final computed value(s) by the network. Note that the size of the last layer depends on the application's objective. For example, if the NN is used for a *classification* objective [87], the output layer may have several nodes, such that each node refers to a corresponding class label. While in *regression* applications [88], the output contains only one node that corresponds to a single score.
An NN composed of a succession of layers is refereed to as *n-layer neural network* with n corresponds to the number of hidden layers, an NN is considered to be *deep* or *shallow* w.r.t. the number n. Such that, the NN is considered as deep if $n > 1$ [75]. Hence, the definition of the *deep learning*: *"A class of*

machine learning techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification." [75].

The objective of every layer in the NN (except the input layer) is to approximate a function f^*, based on the transformation function f (equation 2.1) applied at every node of that layer. The training process corrects the parameters θ for a better approximation of the predictions through a set of learning iterations. Hence, the approximated transformation function of a given layer of the MLP is as defined in equation 2.2.

$$f^*(x) = \delta\left(Wx + b\right) \tag{2.2}$$

where W and b refer, respectively, to the connection weights matrix and the bias vector corresponding to one layer in the MLP.

2.3 Some NN Architectures

Several NN architectures have been proposed to address different tasks. In this section, we present mainly those used in this thesis.

2.3.1 Convolution Neural Networks (CNN)

Because the simple MLP neural networks are not able to process data with several dimensions, the CNN models are used to handle data with more than 2 dimensions. In this section, we describe basic characteristics of the CNN networks. Models using these networks for text matching applications are described in chapter 4. The convolutional network is first used in 1989 by Denker et al [89] and Lecun et al. [90] for handwritten character recognition. The follow-up works in image processing [91, 92, 76, 93] use a large range of CNN architectures. The trend to use the CNN models is due to their hierarchical architecture allowing a multi-level data processing, within two main types of layers: *convolutional* layers, and *subsampling* (more commonly *pooling*) layers. The main characteristics of the convolution network are as follows:

- *3D layering of neurons.* the layers can be of 3 dimensions, *width*, *height* and *depth*. The neurons of a layer are connected to only a few nodes of the layer before ;
- *Local connectivity.* the CNN first creates representations of a small parts of the input data sample, then assembles these representations to handle larger areas. This is performed by enforcing the local connectivity pattern, using different filters for a spatially local pattern recognition;
- *Sharing weights.* since the CNN uses several filters to extract different local patterns from the input image, every filter is slid along the image using the same free parameters. Hence, the number of trainable parameters is reduced, and every filter recognizes a specific pattern with a corresponding weight.
- *Convolution and Sub-sampling.* The convolutional layer shares many weights through the different filter windows, and the pooling layer sub-samples the

output of the convolutional layer to reduce the data rate received from the layer before. A subsampling layer can be a *max-pooling* function or another aggregation function (*avg* or *sum*) that maps a window of the input matrix to a single value.

Every time a convolution layer is applied, the image's size shrinks. Besides, the corner pixels of the image are used only a few number of times during convolution as compared to the central pixels, and this leads to information loss. To overcome this issue, CNN models use a *padding* function which consist on adding some values, usually zeros, to the image borders to fit in the original input shape.

2.3.2 Recurrent Neural Networks (RNN)

Recurrent neural networks are based on David Rumelhart's work in 1986 [94]. A RNN is a network with backward (recurrent) and forward connections between the different nodes. These connections form a directed graph along temporal sequence. The *recurrent* connections bring the information from nodes of the output back to nodes of the input. Figure 2.5 shows an example of a RNN with bi-directional connections (bi-RNN) [1]. A RNN is used for modeling sequence

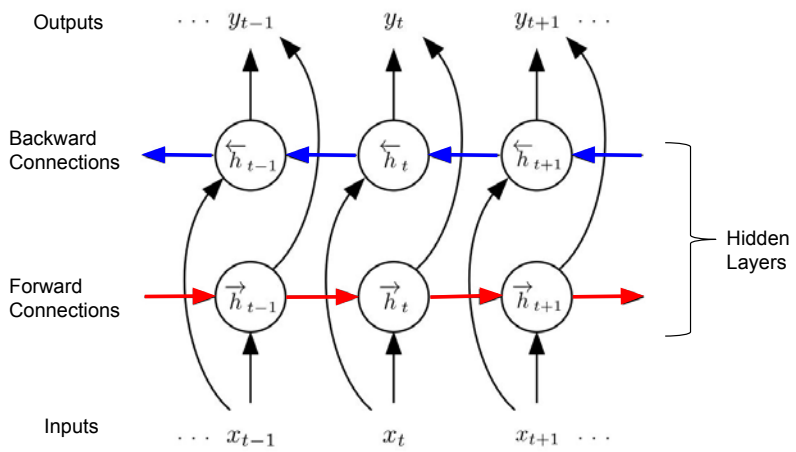


Figure 2.5: Example of a bidirectional recurrent neural network (bi-RNN) [1].

data of different applications, such as speech recognition [95, 96, 1] and text processing [97, 98].

In a RNN, given an input sequence $x =< x_1 ... x_n >$, the hidden layers of the network compute a tensor of hidden state vectors $h = [h_1, \ldots, h_n]$, where every hidden state h_t is then used to compute a corresponding output y_t, as shown in equation 2.3 [98].

$$\begin{aligned} h_t &= \delta\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right) \\ y_t &= W_{hy}h_t + b_y \end{aligned} \tag{2.3}$$

where W_{xh}, W_{hh} and W_{hy} are connection weights corresponding to the input-hidden, the hidden-hidden and the hidden-output connections, respectively. b_h and b_y are bias corresponding respectively to the hidden and the output layers.

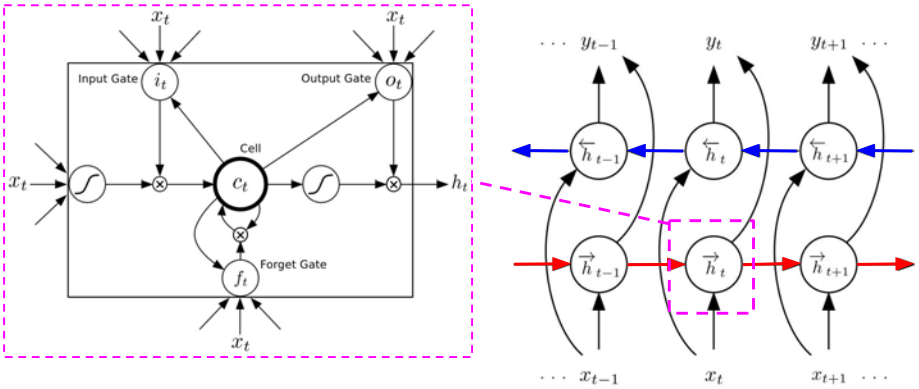


Figure 2.6: Closer view of LSTM cells used in a bidirectional RNN network.

δ refers to the activation function.
In case of bidirectional RNN structure, as shown in figure 2.5, the hidden state h_t is computed in both the forward an the backward directions [99]. Hence, the equation 2.3 becomes as shown in equation 2.4, where $h_t = \left[\overrightarrow{h}_t, \overleftarrow{h}_t\right]$ is the tensor that is computed by the hidden layers.

$$\begin{aligned} \overrightarrow{h}_t &= \delta\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right) \\ \overleftarrow{h}_t &= \delta\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t-1} + b_{\overleftarrow{h}}\right) \\ y_t &= W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y \end{aligned} \tag{2.4}$$

The RNN models can use their recurrent connections to explore the sequential information of an input sequence. However, error signals flowing backward in time tends to blow up or vanish, resulting in a kind of information oblivion by the RNN. To overcome this problem, Long short term memory (LSTM) cells have been introduced [100].
An LSTM cell is a complex node of a RNN with memory. These cells are composed of several gates associated to the activation vector of the neural node. The different gates are regulators that correct and modify the signal being processed by every node, according to the previous information. Hence, every node in the LSTM network has an *input gate*, an *output gate*, and a *forget gate*, thus allow the constant error to flow through the network. A description of the LSTM cells is provided in figure 2.6, where the hidden states h_t is computed following the pipeline functions of equation 2.5.

$$\begin{aligned} i_t &= \delta\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \\ f_t &= \delta\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \\ c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \\ o_t &= \delta\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-}b_o\right) \\ h_t &= o_t \tanh\left(c_t\right) \end{aligned} \tag{2.5}$$

where δ is the activation function of the hidden layer. i, f, o, and c are respectively the *input gate*, *forget gate*, *output gate* and the *cell activation vector*, all corresponding to the current input x_t.

Some variations of the LSTM units do not have one or more of these gates or may have other gates. For example, gated recurrent units (GRUs) [18] do not have an *output gate*. Further more, a bi-RNN architecture using LSTM cells results in a bidirectional LSTM (bi-LSTM) [101]. The latter can access long-range context in both input directions, thanks to the LSTM memory cells and the bidirectional processing of the bi-RNN.

2.3.3 Transformers

The Transformer is a deep learning model introduced by Vaswani et al. [102], used in processing natural language data such as text. A Transformer is designed in a same way than a RNN based solely on attention mechanisms [31] and simple feed forward layers, to handle ordered sequences of text. The main difference between the RNN and a Transformer is that the latter does not take into account the information (words or sequences) ordering, allowing for much more parallelization than RNNs during training [102].

Since their introduction, Transformers have become the basic building block of most state-of-the-art architectures in natural language processing [103]. Since the Transformer architecture facilitates more parallelization during training computations, it has enabled training on much more data than was possible before it was introduced. This led to the development of pretrained systems such as BERT (Bidirectional Encoder Representations from Transformers) [104].

2.4 Neural Models Training

In order to train a neural model, an *objective function*, sometimes referred to as a *cost function*, or a *loss function*, is defined to measure the learning error w.r.t. the desired data. This error represents the difference between the desired output (true labels) and the predicted value. Based on this error, the network is trained by updating its free parameters with the objective of minimizing it. To do so, weight updates are made to continually reduce the error until, either a good enough model (with a minimum error rate) is found, or a specific number of training iterations is reached.

Depending on the application's objective, the model can be *supervised* when true labels are given to adjust the model's parameters; *weakly supervised* when a few, imprecise or noisy true labels are provided for training the model in a supervised manner; or *unsupervised* where no truth information is provided for training. The general training process, in case of supervised applications, can be summarized as follows: first, select a set of training samples, then propagate the inputs over all the network, by computing several intermediate results until reaching the final output layer. The latter provides a value that is then used in order to compute the error rate compared to the true label corresponding to every input sample. Finally, the model parameters are updated w.r.t. the error value.

In order to train deep learning models, we need massive sets of training data. Many traditional lines of research in machine learning (ML) are motivated by training data issue. The hand-labeled training datasets are expensive and time-consuming to be created — often requiring subject matter experts (SMEs) to get accurate true labels.

Application Type	Function	Equation	Description
Regression	Square error	$(y-\hat{y})^2$	Quadratic difference between the prediction and the true label. It assesses the quality of the estimator.
	Absolute error	$\lvert y-\hat{y}\rvert$	A measure of the difference between two continuous variables: prediction and label.
Classification	Square loss	$(1-y\hat{y})^2$	A real-valued error focused on classification which has a strict "yes"/"no" interpretation. It indicates if predictions were correct or not.
	Hinge loss	$\max\{1-y\hat{y}, 0\}$	It considers that when the prediction and the label have the same sign the prediction is true. It is used for "maximum-margin" classifications, such as SVM.
	Logistic error	$\frac{1}{\ln 2}\ln\left(1+e^{-y\hat{y}}\right)$	It converts log-odds to probabilities of correspondence between the prediction and the labels.
	Cross entropy	$-y\ln(\hat{y})-(1-y)\ln(1-\hat{y})$	It measures the average number of bits needed to identify an event using an optimized function for an estimated probability distribution, rather than the true distribution

Table 2.2: List of widely used loss functions per model objective.

In the followings, we describe different types of training directions driven by the data availability. The *supervised*, *weakly-supervised* and *unsupervised* training process are described in sections 2.4.1 and 2.4.2, respectively.

2.4.1 Supervised Training

Supervised learning techniques construct predictive models by learning from a large number of training examples, where each training example has a corresponding true output label called also the *ground-truth*. In supervised learning, neural models are trained to minimize an error value. This value is measured by the difference between all the values predicted by the trained model and the corresponding true label values. Different cost functions compute different errors for the same prediction, and will therefore have a significant effect on the model performance, depending on the task being addressed, i.e. *regression* or *classification*. In this section, we present some widely used error functions.
Given an input data vector X, a true labels vector Y corresponding to X, and a prediction vector $\hat{y} = g(x;\theta)$, where g refers to the NN model architecture, and θ represents all the model parameters. The model predicts an output value $\hat{y}$ for each input x in the training sample. An error associated with $\hat{y}$ is then computed comparing the prediction to the corresponding true label y. This error is computed using a loss function $\pounds(y,\hat{y};\theta)$. This function varies from one model to another. Table 2.2 gives a list of widely used loss functions.
In practice, loss values are estimated for the entire set of different samples in the training/evaluation dataset, and the value to be considered is called *mean error*. This value is computed by averaging the different loss values observed over all the different samples (examples) of the dataset and computed with the same loss function.

2.4.2 Weakly-Supervised and Unsupervised Training

To overcome the problem of truly labeled training data unavailability, several solutions have been used to generate reliable training data. Many methods have found that, unlabeled data when used in conjunction with a small amount of labeled data [105], or using *weak* labels [106], that may be noisy or of lower quality but larger-scale, can produce considerable improvement in learning the model's accuracy.

The recent trend in ML research and deep learning applications tends to use *weak labels* to train finite models, for classification [107] and for regression [108] objectives. The weak labels are automatically generated and computed rather by a pre-trained model [109] or by a model assumed to be effective [110]. In [111], Zhou gives an overview of different weak supervision types.

2.4.3 Unsupervised Training

Differently from the *supervised* and *weakly-supervised* learning methods, the *unsupervised* learning methods aim to train a model in order to predict accurate results such that no true labels are provided. The objective is to directly infer the properties of the dataset density without providing correct answers or degree-of-error for each observation [112]. Compared to supervised methods, the unsupervised learning the error is in general computed w.r.t. the input compared to its predicted representation, in a representation learning process, or by minimizing the distances between elements of the same group, in a clustering objective.

Note that in this thesis our work is not concerned by the weakly-supervised and unsupervised approaches, we propose only supervised models.

2.5 Training Algorithms

Several algorithms can be used to train NN models. In the following, we give a broad definition of the most used ones.

2.5.1 Backpropagation

The *backpropagation* is an algorithm used to train neural networks. It can be seen as an application of a chain computation rule or *chain reminder* that uses a backward differentiation [10]. Rumelhart et al. [94] have proven that this algorithm can effectively train multi-layer neural networks undoubtedly, which was a major catalyst for subsequent research in training neural networks with backpropagation [113, 114, 115]. It is based on the gradient (derivative) of an objective function $\pounds$ with respect to the connection weights W of a multilayer NN. The backpropagation process is applied repeatedly to propagate gradients through all nodes of the NN, starting from the output all the way to the input (backward pass) [10].

Formally, let g be the NN model to be trained. Given an input data sample x, a corresponding true label value y, and the predicted output value $\hat{y} = g(x)$. The backpropagation method aims to learn a mapping function $g : x \rightarrow y$, based on a differentiable objective function $\pounds\left(\hat{y}, y\right)$ that computes the error value at each iteration, to better approximate the true association of every element in the input space X to the corresponding element in the output space Y.

2.5.2 Gradient Descent

Gradient descent, also called *gradient backpropagation*, is one of the most popular algorithms for optimizing neural networks. Gradient descent is a way to minimize the cost function $\pounds(\theta)$ by updating the parameters in the opposite

direction of the gradient $\Delta\pounds$ w.r.t. the free parameters θ. A learning rate α is then used to determine the size of the steps taken to reach a local or global minimum optimized error value. Hence, the parameters θ are updated at each training iteration t as shown in equation 2.6.

$$\theta_t = \theta_{t-1} - \alpha\Delta\pounds(\theta) \tag{2.6}$$

There are three variants of the gradient descent algorithm, which differ in the number of input examples (samples) used to compute the gradient of the cost function $\pounds$. Depending on the quantity of examples given, usually, there is a trade-off between the quality of the parameters update and the time required to perform an update.

- *Batch Gradient Descent (GD).* is computed on all input examples and their corresponding labels, then perform a single update, at every iteration. The GD algorithm can be very slow and even impossible for data sets that do not fit in the memory.
- *Stochastic Gradient Descent (SGD).* differently than the GD, it uses the cost gradient of only one example at each iteration. At each iteration, the model parameters are updated using a learning example drawn at random, which makes the algorithm faster.
- *Mini-batch Gradient Descent (MGD).* combining the GD and SGD, the MGD updates the model's parameters for each minibatch (training subset) of the input data. Hence, the MGD reduces the variance of parameter updates (relative to SGD), which can lead to more stable convergence.

2.6 Over-fitting and Regularization

The over-fitting happens when the model contains more parameters than can be trained with the dataset that is used (*train set*), ie: the model's complexity is not effective to the training data [116]. Hence, the resulting model struggles to generalize or produce accurate results in an unknown dataset (*validation set*).

During training, the weights grow in size in order to handle the specifics of the examples seen in the training data. The large weights make the network unstable, resulting in large differences in the outputs corresponding for minor variation or statistical noise on the expected inputs [117]. In order to avoid over-fitting situations and poor performances when making predictions, we need to update the learning algorithm to encourage the network to keep the weights small and penalize the large weights. This is called *weight regularization* and it can be used as a general technique to reduce over-fitting of the training dataset and improve the generalization of the model. Traditionally, regularization is conducted by including an additional term in the cost function of a learning algorithm. In [118], several illustrative examples on weights regularization have been compared. Several regularization techniques have been suggested in the literature, such as early stopping, dropout, weight decay and curvature-driven smoothing, as described in [80]. In the following, we describe three regularization methods that are widely used in different deep learning applications.

- *Early stopping.* [119, 120] While using this method, the available data samples are divided into three subsets: training, validation, and testing

sets. The error on the validation set is monitored during the training process and when the validation error increases for a specific number of iterations, the training is stopped, and the weights at the minimum of the validation error value are returned.

- *Dropout.* [121, 122] This method consists on randomly omitting, with a defined probability value, a part of the feature detectors (nodes) on each training example. It aims to prevent complex co-adaptations of the different units on the training data. In [122], Hinton et al. have performed an empirical study evaluating several dropout rates in different layers of a NN for images classification, and showed that a dropout values of 0.2 and 0.5 enable to highly reduce the classification errors relative to different existing methods.

- *Batch Normalization.* [123] This method makes the data normalization a part of the model's architecture. It performs the normalization for each training mini-batch. The objective of this method is to improve the training and reduce the impact of the changes in the distribution of network activations, due to the changes in the network parameters. Hence, make the network training converges faster.

2.7 Conclusion

In this chapter, we presented some basic concepts on neural networks and deep learning.

The data availability and accessibility have encouraged researchers to study the current world in different areas. The resulting research work have came up with several training tips and strategies (sections 2.4 and 2.5) to enhance and develop deep learning applications.

In this thesis, we are interested in application of these new technologies in different text matching applications. We describe and discuss several models for representation learning and matching in Part III of this document.

Part III

State of The Art Overview

CHAPTER 3

TEXT REPRESENTATION MODELS

3.1 Introduction

Multiple issues and weaknesses have been mentioned while using the classical BoW representations in different text matching tasks [124, 125, 126], such as the large features dimension, highly sparse representations, lack of semantics and the vocabulary mismatch. Another important issue is related to short queries that can be ambiguous [127]. These as are the major limitations that have encouraged research in NLP to focus on dense representations capable of capturing semantics and context information.

Vector space models have already been used in distributional semantics [128]. Since then, we have seen the development of multiple models used for estimating continuous representations of words in order to overcome the limitations of the BoW representations. Refereed to as *word embedding* (the most used), *distributional semantics* or *distributed representations*, these representations are based in a language model theory [129, 130] to learn to predict words by knowing their contexts. Some of them [51] use neural networks and hence, are called neural network language models (NNLM). The NNLMs, as a variety of distributional semantics models, have shown [131] their effectiveness, in word analogy and semantic relations tasks, compared to Latent Semantic Analysis (LSA) [47, 132]. Bengio et al [51] were the first to propose a neural language model by introducing the idea of simultaneously learning a language model that predicts words based on their contexts. This idea has since been adopted in many studies. Some of the widely used models for distributed word representations in text matching tasks, including NLP and IR applications, are the Word2Vec model [2] and GloVe [9]. The success of distributed word representations has also led to work on learning distributed representations for larger text units, including paragraphs and documents [8]. Other works [133, 134, 135] make use of external semantic resources, in conjunction with the distributed word vectors, in order to learn accurate representations for words and sequences of words. These methods are out of the scope of this thesis, more details about these models can be found in the work of Nguyen et al [136].

The main motivation of using the distributed representations of words as well as of sequences, in text matching applications, is that simple vector calculus

Notation	Description
e	embedding function
E	embedding space
dim	dimension of the embedding space
$\vec{w}_i$	embedded vector of word w_i
$s =< w_1 ... w_{\|s\|} >$	text sequence
c	context window of several words
w_t	word at position t of a word sequence
$\vec{Q}$	query embedded vector
$\vec{D}$	document embedded vector

Table 3.1: Set of notations.

reflect semantic relatedness between words and sequences. For instance, distances between word vectors are semantically significant, since word embedded vectors are learned so that words belonging to the same context have similar vectors [52]. The same applies to sequence vectors [2, 12], where sentences that share semantic and syntactic properties are thus mapped to similar vector representations [12].

In the following sections, we use the notations defined in Table 3.1. We assume that words are the smallest unity of a text, and do not consider the character-based representation models [137, 138]. We detail the main works related to distributed text representations and their different levels of granularity. Specifically, word representation models and sequence representation models are presented, in sections 3.2 and 3.3 respectively. Several text matching models that are based on distributed representations of words and sequences, are presented in section 3.4.1. Finally, in section 3.5, we discuss several issues related to the learning and the use of distributed representations of words and sequences.

3.2 Distributed representations of words

In this section, we present some models for learning distributed word representations. These models are grouped into two different classes [9], methods based on *matrix factorization* and methods based on *local context windows*.

3.2.1 Matrix Factorization Methods

The matrix factorization (MF) approach is used to factor analysis procedures [139, 140]. One of the most used and simpler factorization methods for text representation is the singular value decomposition (SVD) [141]. In text mining, this method is used to handle the semantic aspects of a large text collections, and construct dense representations. Such as the LSI [47], the LSA [142] and the PLSI [143] latent models. These models use a large *term-document* or *term-context* occurrence matrices to capture statistical information of different words in a corpus. These matrices are decomposed using the SVD method, in order to capture the most important latent concepts for indexing huge amount of text data, and hence construct a semantic-based dense representations for words and sequences.

Differently from the aforementioned latent space-based methods, other methods such as, HLA [144], COALS [145] and HPCA [146] use *word-word* co-occurrence matrices, where both rows and columns correspond to words and the entries correspond to the number of times a given word occurs in the context (window) of another given word. Recently, Pennington et al. [9] have proposed the global vectors for word representation, namely GloVe. This model is a word representation learning method that combines the MF technique, using a *word-word* matrix ; with the *local context* window of every word. This method exploits the "global" log-bilinear regression model [147], which is a statistical technique used to analyze the relationship between more than two categorical variables. GloVe leverages statistical information about a text dataset, by training the model only on the nonzero elements of the co-occurrence matrix, rather than on the entire sparse matrix or on individual context windows of a large corpus. It combines the global and the local contexts during training to learn the embedded representations of words. This process is highlighted in the objective function of equation 3.1 [9].

$$\pounds = \sum_{i,j=1}^{V} f\left(x_{ij}\right)\left(e(w_i) \cdot e(w_j) + b_i + b_j - \log x_{ij}\right)^2 \tag{3.1}$$

where x_{ij}, $e(w_i)$ and $e(w_j)$ are co-occurrence number, and the embedded representations of words w_i and w_j, respectively. b_i and b_j are bias values associated to $e(w_i)$ and $e(w_j)$, respectively. V is the dataset vocabulary, and $f(x)$ is a weighting function defined as follows:

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases} \tag{3.2}$$

where α is a defined parameter, such that rare and highly frequent co-occurrences are not overweighted.

3.2.2 Local Context Window Methods

Differently from the matrix factorization methods, the local context-based approach aims to learn word representations based on the occurrence window called *local word context*. The neural network language model (NNLM) [51] is the origin of context-based methods for learning word representations, and it is a probabilistic language model where word probabilities are computed using a neural network architecture. The NNLM simultaneously learns distributed representations *embeddings* for input words, and the probability function for the corresponding context windows using its word vectors. NNLM uses a language model such that, given a sequence $s =< w_1 ... w_{|s|} >$, where $w_t \in V$ is the word at position t. The objective is to learn a function f that computes the probability to get words of a given context window $c =< w_{t-n+1} \cdots w_t >$ of length n in s, using a NN architecture. The NNLM computes the probability of getting a sequence of words knowing a current word w_t, using a NN that takes in input the sparse vector of every vocabulary word.

The NNLM is trained to optimize a loss function $\pounds$ for all the words of the sequence[1] s and their corresponding context windows c. As defined in equation

[1] In practice, all the context windows are considered in all the training dataset C.

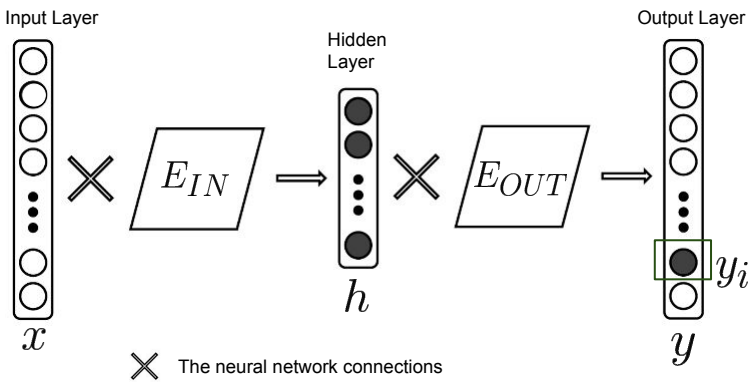


Figure 3.1: A general architecture of the Word2Vec model [2] to learn word mebeddings.

3.3.

$$\pounds(\theta) = \sum_{(w_t,c)\in s} \log P\left(w_t|c;\theta\right) \tag{3.3}$$

where θ refers to all the parameters of the NN used to compute the probability P.

The results obtained by the NNLM model have led Collobert et al. [148] to use the full symmetric context window of a word, in order to learn its representation vector, rather than just predicting its preceding context words. In their model [148], Collobert et al. use a symmetric context window $c =< w_{t-n+1} \cdots w_t \cdots w_{t+n-1} >$. Hence, the context of a word w_t refers to n words before and n words after.

Following the success of previous NNLM methods, Mikolov et al [55, 2] proposed an effective embedding method, called *Word2Vec*, for computing distributed representations of words. Figure 3.1 shows a general architecture of the Word2Vec model, where x corresponds to the NN input vector, h is the hidden layer of dimension dim, and y is the output vector. Two different configurations of this architecture have been adopted in [2]. Specifically, the CBOW and the Skip-Gram architectures are both based on the architecture of the NNLM. Mikolov et al. have adapted several techniques to improve the learning effectiveness and the quality of the outcome representations.

The CBOW model [2] is similar to the model of Collobert and Weston [148], and is trained to predict a word w_t based on the words in its symmetric context c. First, words of the context c are aggregated (sum or average) and fed to the model. After training, the vector representation is provided by the hidden layer h of the NN. Differently from the CBOW architecture, the Skip-Gram model [2] is trained to predict the symmetric context, given the word w_t in the center. In both configurations of the Word2Vec (CBOW and SkipGram), for each word-context pair (w_t, c), the ith element in the output layer of the NN (figure 3.1), and is computed as shown in equation 3.4.

$$y_i = e_{IN}(c).e_{OUT}(w_t) = E_{IN}.\vec{c}.E_{OUT}.\vec{w}_t \tag{3.4}$$

where $E_{IN}, E_{OUT} \in \mathbb{R}^{|V|\times dim}$ are the NN connection weights related to the

hidden layer h of size dim, and consist on the embedding spaces. $\vec{c}$ and $\vec{w_t}$ are respectively the context and the current word vectors, with $\{\vec{c}, \vec{w_t}\} \subset \mathbb{R}^{|V|}$.

In case of the CBOW configuration, w_t is the central word to be found; and c is its corresponding context having n words before and n words after. Vectors of these words are aggregated in a single input vector $x = \sum_{w \in c} \vec{w}$ and fed to the NN (figure 3.1). While in the Skip-Gram model, the objective is to predict a word w_j which consists on a word from the context window of the central word w_t. Hence, for each central word w_t, the network will iterate $2 \times n$ times with the same input $x = w_t$ in order to predict all the words $w_j \in \{w_{t-k}, ..., w_{t-1}, w_{t+1}, ..., w_{t+k}\}$ of the context window [149]. Each predicted w_j is expected to be of the same semantic context as the previous predicted words.

Following several results and analysis of the context-based distributed word representations, some authors have been interested in the adaptation of the Word2Vec model [2] to the task being addressed. For instance, the cross-lingual embedding models [150, 151, 152], the DESM [153] composed of the two different embedding sub-spaces IN and OUT (figue 3.1), and the relevance-based embedding model [154] for query expansion. In the scope of this thesis, we have made a large use of the previously proposed embedded word representations, specifically Word2Vec [2] and GloVe [9], in order to exploit the semantic aspects of words and sequences in a text.

3.3 Distributed Representations of Sentences

The distributed word representations have been the basis of several works on text matching applications, such as question answering [155, 156, 157] and document retrieval [158, 159, 160], by exploiting the word-level semantic relatedness between words within different contexts.

Several models have been proposed to construct embedded representations for sentences and documents. We can divide the different models into two main categories [136], according to the approach which is used to generate the embedding of the input text: 1) *aggregated* [161, 162, 163, 164], where the proposed models construct the embedded vector of a text based on the component word representations, and apply aggregation functions to build the sentence representation. 2) *non-aggregated* [8, 12, 165, 166, 167], in this class, models directly obtain a representation vector of a text sequence without using a summation-based combination of component word vectors, and a non linear method is used.

3.3.1 Aggregated Representations

Models of this class use a linear function that combines word vectors of an input sequence to construct its corresponding representation vector. This class includes models that are based on the simple *Averaged Word Embedding* (AWE) approach [168, 150, 158, 13]. The AWE method consists on using the sum or average of the embedded vector representations of component words in a given sequence, to produce a global vector representation. Such that, for a given sequence words $s = \{w_1 ... w_m\}$, the corresponding embedded representation

$e(s)$ is computed as in equation 3.5.

$$e(s) = \frac{1}{\sum_i \beta_i} \sum_{w_i \in s} \beta_i e(w_i) \tag{3.5}$$

where β_i is the weight of word w_i, and $e(w_i)$ its embedded vector. Note that in case of a simple average function is used $\beta_i = 1$, $\forall w_i \in s$, and hence $\sum_i \beta_i = m$

Several models are based on equation 3.5 to estimate embedded representations for word sequences. For instance, Kenter et al. [161] have proposed the Siamese Continuous Bag of Words (Siamese CBOW) model. This method aims at learning embedded word vectors and aggregating them using an average function, in order to construct embedded representations for input sentences. A cosine similarity function is then used to match the different sentence vectors. During training, the model optimizes the word embeddings to better compute the whole sentence representations. The Siamese CBOW [161] is based on a supervised set up, and learns to predict sentences occurring next to each other in the training dataset. Differently from the supervised Siamese CBOW model [161], Hill et al. [162] have proposed the Sequential Denoising Auto-Encoder[2] (SDAE), built with an LSTM-based encoder-decoder architecture, and which is a supervised method for learning sentence embedding from an unlabelled dataset. This model is trained to optimize the embedded word vectors. The final sentence representation is computed by aggregating component word vectors, using a linear combination. The SDAE model first introduces some noise by swapping some bi-gram word vectors of an input sequence, then is trained to reconstruct the original sentence.

Without using a neural architecture, Arora et al. [163] have proposed a strong new baseline for sentence embedding learning. This method consists on a simple weighted average of the word vectors of a given sequence, as described in equation 3.5. The averaged vector representation is then modified using a principal component analysis (PCA) [170] or the singular value decomposition (SVD) [141] method. The authors first construct all the sentence vectors by averaging their word embeddings, then compute a singular vector for every sentence, in order to optimize the different representations and reduce the sentence vocabulary space. Another similar approach is proposed by De Boom el al. [164], and that is also based on averaging weighted word vectors of a given sequence. Given the pair of sentences to be matched, the model first computes initial averaged embedding vectors for both of them using same word weights. After that, the weights are updated using a gradient descent process, which is expected to minimize the error corresponding to similar sentences, in order to learn better coefficients for the average function.

The simple averaged embedding treats all words with the same importance in the combination. Although some works [150, 158] have considered terms weighting (equation 3.5), Zamani et al. [165] have argued that aggregating word vectors of a long sequence could result in imprecise representations of the corresponding semantic content. Links such as, the dependencies and similarities between different words, sequences, and paragraphs in a text are not

[2] An autoencoder [169] is a neural network that learns to learn data representation by reducing the dimension of the inputs, mainly it is made of two parts, the encoder that learn to map the input to a lower dimension representation, and the decoder that tries to reconstruct the input data by decoding the representation learned by the encoder.

taken into account. To cope with this limitation, non-aggregated representation models have been proposed.

3.3.2 Non-Aggregated Representations

Non-aggregated representation learning approaches do not use AWE functions. These methods construct a latent vector for every sentence using a function that learns to map the initial word representations (pre-trained embeddings or sparse one-hot vectors) to a latent representation of the whole sentence.

One of the most popular models for computing sequence embeddings is the *ParagraphVector* of Le [8] which is derived from the previous work *Word2Vec* [55] for learning embedded word representations. The ParagraphVector model considers a paragraph[3] as an atomic unit instead of a combination function of its component words. Hence, it extends the Word2Vec model's architecture with an additional input token dedicated for learning paragraph embeddings. It is based on maximizing the average log probability of getting an in-context word, given the other words of its context and the sequence containing them. Differently from this model, the *Skip-Thought* model [12] preserves the semantic compositionality principle[4] while learning distributed representations of input sentences. Skip-Thought consists on an encoder-decoder neural network [18, 31]. Its encoder is a recurrent network (RNN) that produces a vector representation of the source sentence and the decoder is another RNN that sequentially predicts the words of adjacent sentences. The underlying assumption [12] is that, in the content of a sentence, anything that leads to a better reconstruction of neighbouring sentences is also essential to the representation of the current sentence.
In [172], Ai et al. propose the *enhanced* paragraph vector (EPV-DBOW) by adapting the original ParagraphVector from [8] to IR tasks. It includes three modifications of the original ParagraphVector: (1) the word frequency in the collection, used in the ParagraphVector, is replaced in EPV-DBOW by the document frequency. (2) to avoid over-fitting short documents, an L2 regularization parameter is used in EPV-DBOW, and takes into account the document length. (3) EPV-DBOW first uses the document to predict the target word, then the target word to predict its context in that document. Another model [167] is proposed by Logeswaran and Lee, and is called *Quick Thought* (QT). QT learns the sentence representations while learning to predict the context in which a sentence appears, among other contrastive sentences. QT uses an encoder-decoder architecture, where the encoder computes an embedded representation of the input sentence, then the decoder attempts to find the embedded vectors corresponding to its component words.

Based on a classical language model, Zamani and Croft [165] have proposed a theoretical framework for estimating query embedded representations based on the individual embedded vectors of all vocabulary words. Specifically, the model provides a way to learn query embeddings using a softmax or sigmoid transformations of the vocabulary word vectors, to compute a maximum like-

[3]The paragraph corresponds to any sequence of words, such as sentences, passages or a whole document.

[4]Principle of Semantic Compositionality [171] *"is the principle that the meaning of a whole is a function only of the meanings of its parts together with the manner in which these parts were combined".*

lihood probability estimating the query representation vector. Wu et al. [173] have proposed the word mover's embedding (WME) model for constructing distributed representations for different length sequences and documents. Extending the word mover's distance (WMD) [174], the WME model considers the distance between two documents as the minimum transportation cost between the two documents. This model learns document representation vectors by generating a random document from which the distance is calculated using the WMD. This process is repeated *dim* times for every document in the dataset resulting in a vector representation of dimension *dim*.

Recently, Park et al. [175] have proposed a supervised paragraph vector (SPV) for learning distributed representations for words, documents and class labels. The original ParagraphVector model [8] generates a single representation for a given sequence which is then used for all tasks. Hence, Park et al. [175] assume that different tasks may require different representations. The objective is to propose a task-specific representation learning algorithm which extends the paragraph vector model to a supervised method. The SPV model uses class labels along with words and documents, and obtains corresponding representations with respect to the particular classification task, enabling to obtain the representations of words, documents, and class labels. SPV [175] uses a simple MLP network of three layers. Each neuron in the input layer refers to one word. The main difference of the SPV from the ParagraphVector [8] is that the documents class label information is used in input. The learning process of the ParagraphVector assumes that a document vector can be derived by predicting sequences of its constituent words. While the training of the SPV model assumes that a class vector can be derived by predicting sequences of its content.

3.4 Text Matching Using Distributed Representations

The distributed representations of words and sequences are used in order to overcome the weaknesses of the classical BoW representations, and the exact-matching models. They enable to use a non-exact matching where distances represent semantic links between the matched words or sequences.

We identify two main ways of using distributed representations of words and sequences: (1) models for *direct matching* including classical and neural models that leverages the distributed representations of words and sequences in order to better match texts; (2) models for *query expansion* that leverage the embedded word representations in order to find better candidate words for expanding the user query. In this work, we focus more on models of the class (1), and describe in more details several models of this class.

3.4.1 Direct Matching

We refer with *"direct matching"* to all models using distributed representations for the purpose of text matching except expansion-based methods. Direct matching methods, include: (a) classical text matching models that use either BoW representations [57, 129], word embedding representations [176, 174], or sequence embedding representations [172, 177]; (b) neural text matching models

[16, 178, 179, 180, 21], where most of the proposed models use the distributed representations in the same way. First, construct distributed representations for the input texts being matched, based on vectors of their component words, then a NN with a succession of different layers extracts interaction characteristics and computes the similarity matching score. The neural text matching models are detailed in chapter 4. In this chapter, we consider classical models (a) and we focus on models involving distributed representations of words and sequences. The objective is to analyze how these representations have been considered.

Embedded representations of words can be incorporated in different ways in classical text matching models. One way is to represent input texts (e.g. queries and documents, questions and answers) as sets of their word vectors (BoV), then integrate the word representations into existing matching models. For example, the neural translation language model (NTLM) [181] leverages the neural word embedded representations to enhance a classical translation model [182] for query-document matching. In the NTLM model, a probability $P(q_i|D)$ that a query term q_i appears in a document D is considered as a translation process where the probability $P(q_i|d_j)$ enables to get the term q_i via a term d_j of the document. ie: translate every term in the input query to a given term in the document, as described in equation 3.6.

$$P(q_i|D) = \sum_{d_j \in D} P(q_i|d_j) \times P(d_j|D) \tag{3.6}$$

where $P(d_j|D)$ is the probability that the word d_j appears in the document D. The probability $P(q_i|d_j)$ denotes the translation process and is computed using the corresponding word vectors as follows:

$$P(q_i|d_j) = \frac{\cos\left(\vec{q_i}, \vec{d_j}\right)}{\sum_{w \in V} \cos\left(\vec{w}, \vec{d_j}\right)} \tag{3.7}$$

where w is a word from the vocabulary space V. $\vec{q_i}$ and $\vec{d_j}$ are respectively a query word and a document word vectors.

Another way to use the embedded word representations in text matching is to assign importance weights to word vectors of the input text sequences. For instance, Zheng and Callan [183] have proposed a model for weighting query terms based on their distributed representations. For a given query Q, the authors first compute a features vector $\vec{t_i}$ associated to every word $q_i \in Q$. Such that, $\vec{t_i}$ is the difference between the word vector and the mean vector of all the query words. As shown in equation 3.8.

$$\vec{t_i} = \vec{q_i} - \vec{Q} \tag{3.8}$$

$$\vec{Q} = \frac{1}{|Q|} \sum_{j}^{|Q|} \vec{q_j} \tag{3.9}$$

where $\vec{Q}$ and $\vec{q_i}$ are the averaged query vector and the vector of a query word, respectively. The model then uses an *l1-norm* regularization method, in order to learn an importance weight for the features vector $\vec{t_i}$ of every query word. First, a true weight r_i is estimated, and represents a binary score corresponding for every query word based on relevance judgments. Then, a new weight is

computed based on the true weight r_i. The weighted key words of the query are then used on the Indri[5] query language model to rank documents of the collection.

Ganguly et al. [159] propose a generalized language model for text matching based on embedded word vectors. They assume that, in a query-document matching task, the query words that are not used in relevant documents pass through a noisy channel that gives them a new lexical and semantic form. This transformation is expressed using semantic similarities between word vectors. The relevance score is computed with a linear combination of the different word transformations, using words from the document and the collection. Guo et al. [14] proposed a document-query matching model based on cosine similarities between all the words in the query with all the document words. The matching process is considered as a non-linear transport problem from all document terms to all query terms. Hence, the relevance score is computed based on the transport flows from the document to the query.

Word representation models often have poor performance when the matching is performed across all the documents of the collection [184]. When it comes to classical matching models using word embeddings, the analysis made by Mitra et al [184] have shown that good results in re-ranking tasks do not imply automatically good performance on large document collections. In order to overcome this limitation, more complex processing is needed to represent the information contained in a text sequence. One possible strategy to construct an embedded representation for a whole text sequence, is to derive a dense vector from the embedded vectors of its component words. In IR, the query and the document can be compared based on their embedded word vectors using a variety of similarity measures, such as the cosine similarity function [16, 185]. Nalisnick et al [153] and Mitra et al [184] have used an AWE method to construct query and document vectors, such that individual words are represented with the Word2Vec [2] embedding model. They analyzed the characteristics of the Word2Vec representations for estimating the relevance of a document according to a query. They highlighted that it is more appropriate to calculate the IN-OUT similarity between words of the query Q and words of the document D. In other words, the query terms are represented in the E_{IN} space and the document terms in the E_{OUT} space (equation 3.4). The relevance of the document w.r.t. the query is then computed using an averaged cosine function of equation 3.10.

$$\mathrm{DESM}(Q, D) = \frac{1}{|Q|} \sum_{q_{i,IN} \in Q} \frac{\vec{q}_{i,IN} \vec{D}_{OUT}}{\|\vec{q}_{i,IN}\| \, \|\vec{D}_{OUT}\|} \tag{3.10}$$

$$\vec{D}_{OUT} = \frac{1}{|D|} \sum_{d_j \in D} \frac{\vec{d}_{j,OUT}}{\|\vec{d}_{j,OUT}\|} \tag{3.11}$$

where $\vec{q}_{i,IN}$ and $\vec{d}_{j,OUT}$ are respectively the vector representation of the query word q_i in the embedding space E_{IN} and the vector representation of the document word d_j represented in the embedding space E_{OUT}.

Ai et al. [177] have analyzed the usefulness of the ParagraphVector model [8] in classical text matching models. Unlike in [172], They showed that integrating

[5] https://sourceforge.net/p/lemur/wiki/The%20Indri%20Query%20Language/

the ParagraphVector model with a traditional LM approach produces unstable performance and limited improvements. Another model that uses a LM is proposed by Peters et al. [186]. In their model, the authors used a bidirectional language model bi-LM [187], that combines the forward and backward language models to jointly maximize the likelihood on both directions. Such that, given a sequence s, the bi-LM of s is computed by combining the likelihood of the forward and the backward contexts of every word in s, as shown in equation 3.12.

$$p(w_i|s) = \sum_{i=1}^{N} \log p(w_i|w_1, \ldots, w_{i-1}; \overrightarrow{\theta}) + \log p(w_i|w_{i+1}, \ldots, w_{|s|}; \overleftarrow{\theta}) \quad (3.12)$$

where $\overrightarrow{\theta}$ and $\overleftarrow{\theta}$ are the LM parameter in the forward and backward directions, respectively. Experimental results have shown that the bi-LM combined with the pre-trained word embeddings outperforms the original bi-LM [187] which is based on the character-level embeddings.
Recently, Al-Sabahi et al. [188] proposed a modified BM25 model for document summarization using word embeddings. Namely MBM25EMB, this model extends the BM25 model [57] with word embeddings to build sentence vectors from their word vectors. Documents are then represented as sets of their sentence vectors, that are computed by averaging their weighted word embeddings. The parameters *tf* and *idf* of the original BM25 [57] model are computed differently in the embedding space. Specifically, the frequency (*tf*) of a given term in a sentence is measured by the maximum cosine similarity value between this term and all other terms in that sentence. The sentences are then weighted according to the MBM25EMB model in order to construct the summary of a long document. The experimental results show that the MBM25EMB model performs comprehensively better compared to different text summarization state-of-the-art methods.

3.4.2 Query Expansion

The query expansion process [189] is a reformulation technique that aims to enrich the query content with additional words, usually taken from the vocabulary. The added words, called *expansion candidates*, are supposed to help the matching model to find relevant documents and resolve the ambiguity of short queries. When the embedded word representations are used for query expansion, instead of matching the query and the document directly in the latent space, word vectors are used to find good candidates for expanding the user query. The selection of candidate words is general is based on a global vocabulary, where similarities between word vectors are used to select the right terms, then the extended query is used to retrieve documents using a classical IR model. Several methods [160, 190, 191, 165] have been proposed for estimating the relevance of candidate terms to the query. Most of these methods share a common procedure: first, every candidate term is compared individually to each term of the query using their vector representations, then the scores are aggregated for each candidate. Top k candidate terms are then added to the final query. Some models [192] can use additional filtering methods, such as the co-occurrence matrix of the selected candidates with the query words.

In the context of this thesis, we do not propose expansion-based models. The objective of this paragraph is to highlight the query expansion approach as a particular use-case of the embedded word representations in text matching applications.

3.5 Issues Related to Distributed Representations

In this chapter, we described several models for distributed text representation learning, namely at the word granularity level [193, 55, 146, 9] as well as at the sequence level, that can be either a sentence [8, 12, 165] or an entire document [8, 133, 173].

Using distributed representations in text matching models leads to several questions. First of all, the training of these representations is such an important step. The main question is about using pre-trained representations or train the embedding models on our own corpus. Several works [13, 183, 165] are satisfied with using publically available pre-trained embedding models, such as Word2Vec [55] and GloVe [9]. The common argument is that the pre-trained representations are learned using sufficiently large corpora and the vocabulary is sufficiently rich. Besides, the fact of training a distributed representation method is in itself a complicated task given the number of parameters to be tuned, depending on the embedding model that is used, such as the word context window size, the embedding space dimension, the algorithm to be used (Skip-gram, CBOW, GloVe, ...). In general, a context window size is taken between 5 and 10 as suggested by Mikolov et al.[55, 52] and the embedding dimension varies from 200 to 400 where the most used dimension is 300 ; Other authors prefer to train the representation models in the target corpus [159, 14, 191], in a general large text collection [176, 194] or more specifically in a domain specific corpus [195, 15]. This option leads to learning new embedding spaces, and have to deal with the issues related to the embedding training. More specifically, the dataset pre-processing and the architecture of the NN that is used to learn these representations. In [196], it was found that the data pre-processing, such as word lemmatization, before training word embedding models is effective, and leads to capture semantic similarities, especially for rare words. Frej et al. [186] presented a detailed empirical study of how the choice of the neural architecture (e.g. LSTM, CNN, or self attention) influences both, the end task accuracy and the qualitative properties of the learned representations.

Second, another important issue is related to rare words in the corpus. Indeed, to train the word embedding models [55, 9], a minimum word frequency is used. Hence, rare words in the dataset are not present in the learned space. These words, called *out of vocabulary* (OOV) words, cause a problem when they are used in the text sequences being matched, such as a user query or a question, and/or in relevant results, such as documents and answers. To solve the OOV words problem, some methods [13, 153] tend to simply ignore them, while other methods [176, 14] consider the these words as an important matching signals, and define a specific token for every OOV word. These tokens are represented using a random embedding vector and used in the matching process.

3.6 Discussion

The distributed representations of words and sequences have impacted several works in text matching applications. These representations have improved research results when used in some classical models [159, 14, 191, 13, 183, 165], but the performances are limited. Several studies [197, 198] have evaluated the impact of using distributed representations of words in different text matching tasks. These studies have argued that word embeddings do not solve lexical ambiguity problems, such as polysemy, since all the meanings of the same word are represented by the same vector, regardless of its different contexts. Besides, context criteria alone are not sufficient to calculate the approximation or matching characteristics of two different texts, in text matching applications. The similarities computed with these representations may lead to an inappropriate exploitation of certain semantic relatedness. For example, Mrkšić et al. [199] have shown that the word *"cheaper"* is found in words closer to the word *"expensive"*, using the representation vector GloVe [9], in this case, these similarity feature is not appropriate for finding words to expand a user query that is for example about *"cheap apartments for rent"*.

Although recent models [200, 201] consider contextual word representations, such as EMLo [187] and BERT [104], but these representations use a task-specific architectures that include the pre-trained representations as additional features [187], or require fine-tuning [104], in the target corpus to well perform in the task being addressed. Analyses made by Tenney et al. [202] and Qiao et al. [203] have shown several limitations related to the use of these representations. For instance, in [202], the empirical analysis have shown that contextualized embeddings outperform their non-contextualized counterparts better on syntactic tasks in comparison to semantic tasks, suggesting that these embeddings encode syntax more so than higher-level semantics. In [203], Qiao et al. have evaluated the performance of the contextualized word representations learned with the BERT model [104], and showed the effectiveness of these representations in a question-answering focused passage ranking tasks, and the interaction-based sequence matching model. However, this analysis [203] have revealed that on TREC ad-hoc ranking, the evaluated ranking models using BERT representations, even further pre-trained on a large ranking dataset, perform worse than classical learning to rank and several neural text matching models that are pre-trained on user clicks. Furthermore, in [200] the contextual embeddings yield great ranking performance improvements, but they come with a considerable cost at inference time when incorporated in a document retrieval model. Such that, the classical word embedding, in particular GloVe [9], is shown to be more effective in terms of time than the BERT [104] and ELMo [187].

Distributed representations of sequences [8, 133, 173, 12, 165] are also limited in terms of performance. The exploitation of these representations with classical models [172, 153, 184] helped sometimes improving the results. However, the latent features of the sentence vectors are not explicit and do not refer to concrete information about the dataset. Unlike the classical tf-idf signals where every feature refers to a significant statistical aspect about words and documents of the collection, such as the word frequencies, the discrimination discrimination, importance weights, and so on. In [177], Ai et al. have analyzed the usefulness of the ParagraphVector model [8] in classical IR models. They have shown that this sequence embedding model is not suitable for represent-

ing long documents, since it tends to over-fits short documents, which leads to privileging short documents when the ParagraphVector representations are used in a retrieval model. Besides, the AWE-based sequence representations is not suitable for representing the semantic information inside long sequences (paragraphs and documents), and can produce an insignificant vector representation due to aggregating a large amount of semantic word representations [165].

In order to cope with the limitations of distributed representations of words and sequences used in classical IR models, it would be preferable if the matching model could automatically extract matching signals, and then combine them, in a way that structural and semantic information could be leveraged.

3.7 Conclusion

In this chapter, we have discussed several text representation models that are used to represent single words and sequences. Along this chapter, we have seen how text representation approaches have evolved in order to better represent different texts. In particular, the embedded word vectors have been used in order to leverage non-exact matching information. Different models for embedding sequences of words and whole documents have been proposed. The aim of these models is to better represent semantic information in a word sequence. However, these representations have shown limited performances when used with classical matching models, and more powerful methods are needed in order to build text representations and better perform the matching function.

CHAPTER 4

DEEP LEARNING IN TEXT MATCHING APPLICATIONS

4.1 Introduction

Neural networks are known for their ability to build, in a latent space, distributed representations that are able to capture semantics of different types of information (e.g. images, sound, text). In recent years, deep neural networks have led to remarkable progress in several research areas such as speech recognition, computer vision, and text mining tasks, including several IR tasks [11]. In text processing, the neural models are used to construct distributed (embedded) representations of words and sequences, as well as to perform the matching process.

We have seen, in chapter 3, that the distributed representations of words are not capable of solving several linguistic issues such as, words polysemy [197, 198], and performances are limited document ranking due to several factors related to the training of these representations [204]. Although, the distributed representations of sequences and documents [8, 133, 173] have been proposed, these methods struggle to handle the semantic information when comparing different texts [177]. The text representations are generally learned independently from the matching task. The recent interest in text matching [31, 168, 179, 166] tends to use a set of neural approaches to learn end-to-end models that better perform the task being addressed. In these models, features are automatically learned using a NN structure and no (or few) features engineering is needed. This differs from the models described in the previous chapter 3 that do not rely on relevance characteristics for learning the matching function. The deep neural networks (DNN) are used in matching applications in order to overcome some issues related to the classical learning to rank (LTR) framework [61, 205], where an important human effort is needed in order to prepare a set of matching features. Hence, the basic objective of using DNNs is to enhance those methods and reduce the human intervention, such as the features engineering, on different ranking systems.

In this chapter, we will first discuss a set of basic aspects of machine learning and their use in text matching applications, mainly the LTR method and some related issues (section 4.2). We will then discuss the use of DNN models in different text processing tasks. Specifically, we will present (section 4.3) a set

Notation	Description
X	input space of a model
x_i	one input example
Y	output space (true labels)
y_i	an output label corresponding to an input x_i
Δ	labeled dataset containing pairs (x_i, y_j)
Φ	a matching model (function)
F	features space (set of characteristic functions)
$\pounds$	loss (objective) function
φ	representation function

Table 4.1: Set of notations.

of deep models proposed for text matching applications. Furthermore (section 4.4), we will discuss the main issues related to the use of DNN architectures in text IR and different text matching applications in general. To do so, let us consider the notation previously defined in table 3.1 and extend it with some additional elements that are needed in this chapter.

4.2 Machine Learning for Information Retrieval

Machine learning (ML) has been used in IR applications since the 1980's [206]. In this section, we give a weaver description of the way machine learning methods have been used in IR for ranking.

In general, ML algorithms are used in two main ways in IR: (1) for text representation learning [55, 12], where the input representations are optimized during training. (2) for matching [207, 185], where a ranking function is learned from handcrafted features. Most ML models for IR are focused on learning ranking functions [208, 209, 210, 211, 205, 212], and are refereed to as *Learning To Rank* (LTR) models.

Let F be a feature space that contains a set of predefined feature functions (*tf*, *BM25*, document's section...)[1]. Hence, in the LTR framework, the input space corresponds to $X \subset \mathbb{R}^{|F|}$, where every element $x_i \in X$ is a vector of weighted features. This vector gives weights, computed by the different feature functions in X, relatively to the text sequences being matched (e.g. query-document or question-answer). Let $y_i \in Y$ refers to the true relevance label (e.g. relevance judgment). The LTR model is trained to optimize the function Φ, called *hypotheses*, in order to learn the parameters θ to better combine features of F w.r.t. every input x_i.

In IR, the input space X contains features computed to different text pairs (e.g. document-query), and the true label y_i corresponds to the relevance judgment.

[1]Some features used in the LTR dataset LETOR [213]

4.2.1 LTR Algorithms

Literally, all the existing LTR methods can be grouped [207] into three approaches: the *pointwise* approach, the *pairwise* approach, and the *listwise* approach. In the following, we consider the query-document matching task and describe the different LTR algorithms accordingly. To do so, consider a query Q and a set of candidate documents $\{D_1, ..., D_m\}$. Let Φ be the LTR model to be trained, and Y the output space.

- *Pointwise.* In the pointwise approach [214, 215, 216], the model's input is a features vector x_i corresponding to a pair (D_i, Q), and $y_i \in Y$ is the relevance label that the model Φ is trained to approximate, such that $\Phi(x_i) = y_i$.

  Note that the pointwise approach does not consider the interdependency among the returned documents [61], and thus the position of a document in the final ranked list is invisible to its loss function.

- *Pairwise.* In the pairwise approach [217, 210, 218], Φ takes a pair of features vectors, x_i and x_j corresponding to (D_i, Q) and (D_j, Q), respectively. Such that, D_i is more relevant to Q than D_j. Hence, the model Φ is trained to learn a *preference* function $f(x_i) \succ f(x_j)$, where $\succ$ refers to the *preference* and f computes the relevance score of a document based on its features vector.

  In this case, the loss function measures the inconsistency between the truth *preference* and the model's *preference*, and the objective is to rank the relevant document better than the irrelevant one. However, this approach processes document pairs related to the same query independently. Such that, for a set of features vectors $\{x_i, x_j, x_t, x_z\}$ corresponding to a set of documents $\{D_i, D_j, D_t, D_z\}$ related to the same query Q, where D_i is preferred over D_j, and D_t is preferred over D_z, the pairs (x_i, x_j) and (x_t, x_z) are handled independently and we can not figure out what is the preferred document in (D_i, D_t).

- *Listwise.* The listwise models [209, 219, 220] consider the entire set of documents $\{D_1, ..., D_m\}$ related to Q. Hence, the model's input corresponds to a set of feature vectors $\{x_1, ..., x_m\}$. The model Φ is then trained to approximate the true labels $\{y_1, ..., y_m\}$ corresponding each one to a relevance score associated to every document D_i w.r.t. the query Q.

The aforementioned LTR algorithms differ in their learning objectives, and every objective function takes into account the application's purpose.

4.2.2 Related Issues

The LTR models, have already achieved great success in many IR applications [214, 215, 219, 210, 208, 211, 212], mainly in the modern Web search engines like Google[2] or Bing[3]. In the followings, two main issues to cope with when designing LTR model:

[2]http://google.com
[3]http://bing.com

- *Features engineering.* Another aspect is the feature selection and filtering. These features can help improve the efficiency of training the LTR models [221]. However, their preparation is a difficult process, since they are hand-crafted. Indeed, the major obstacle in LTR is the availability of high quality relevance features. Most existing LTR models rely on hand-crafted features. These features are time consuming, since their construction involves an important human efforts. Specifically, one needs to, first, select a set of important features, then compute their weights (manually or automatically) for every example in the dataset. This process is often data-specific, and could require domain experts in order to perform the features selection in some cases (ex. describing scientific documents). Although several studies [222, 221] have proposed different automated feature selection methods, the features engineering is still an expensive step and requires a complicated procedures.

- *Training objective.* In [61], Liu et al. have discussed several issues about the objective functions used in different LTR algorithms. In fact, the LTR methods for IR use ML techniques in order to perform the ranking of relevant documents. However, the loss functions that are used by these models, mainly the pairwise and listwise approaches, take into account only part of the ranking objective in IR. To deal with this limit, Liu et al. [61] suggested to adopt loss functions based on the IR true evaluation measures. Such that, this true loss for ranking could be defined at the query level and consider the position of relevant results. The statistical consistency further discusses whether the minimization of the expected risk, defined with the loss function, can lead to the minimization of the expected risk defined with the true loss function.

To cope with these issues, particularly the features engineering, it would be of a great value if the LTR ranking model could, in one hand, automatically learn the useful relevance features, and in the other hand, learn the relevance function which is capable of accurately identifying the relevant items within a diverse and a noisy dataset.
Deep learning models [77, 76, 223] provide a set of powerful computational methods that could give more potential for learning several complicated tasks in text matching, than the traditional shallow models. Essentially, these models enable to learn efficient abstract representations from raw data [224]. Such that, input data samples are represented by distributed structures, where the real values refer to weights of latent features that are automatically captured by the NN. Hence, these models can be used as alternatives in order to avoid the problems of the feature engineering in the traditional LTR framework. Due to their potential benefits, and along with the expectation that similar successes with deep learning could be achieved in IR [225], DNNs are used to develop several deep models to handle different tasks in text representation learning and matching.

In the following section, we will discuss how deep models are used to handle text matching applications, and present a set of deep learning models for text matching applications.

4.3 Deep Learning for Text Matching

DNN models have been used in diverse IR and NLP tasks. This interest is motivated mainly by the accurate learning of distributed representations, such as the distributed vectors of words [55, 9], and the structured arrangement of them [8, 12] for natural language expressions such as sentences and documents, and effectively utilizing these representations in matching tasks [184, 185, 178, 226]. Several NN models have been proposed to model end-to-end text processing tasks. These models have been discussed in multiple tutorials and surveys [227, 228, 11], where the neural models use neural networks order to automatically extract different matching signals and better combine them. These models are used in several text matching applications, for instance:

- *Ad-hoc Search*, such as document retrieval [180, 110, 179] and passage retrieval [229].
- *Question Answering* applications, such as answer sentence selection [230, 231, 232] and community question answering [233, 234].
- *Classification* tasks, such as document classification [235, 25] and paraphrase identification [168, 236]

The different neural models for text matching applications are divided into two main groups [178], specifically *representation-focused* models and *interaction-focused* models.

4.3.1 Unified Model Formulation

Before describing models of the different groups, let us first define a unified model formulation, adapted from [11], and that we further use to describe different neural matching models. Consider the framework described in figure 4.1, where a matching model is composed of two main parts: the *representation* part builds distributed representations of the input sequences, and can use several models described in section 3.3; the *matching* part compares the representations built in the *representation* part.

We first define the input space of a neural model for text matching as $X = S^{(Q)} \cup S^{(D)}$, where $S^{(Q)}$ is the generalized query space, containing the first sequence of the pair of sequences to be matched (e.g. query or question); $S^{(D)}$ is the generalized document space containing the second sequence of the pair (e.g. document or answer). In the input space X, for every generalized query sequence $s_i^{(Q)} \in S^{(Q)}$, we define $T_i = \left\{ s_{i1}^{(D)} \cdots s_{in_i}^{(D)} \right\} \subseteq S^{(D)}$ a set of n_i sequences. Let $\mathbf{y}_i$ be a set of true labels corresponding to the input example $s_i^{(Q)}$ and the set $\mathbf{y}_i = \{y_{i1}, \cdots y_{in_i}\}$ of sequences. Such that, y_{ij} is the relevance label corresponding to the input $s_{ij}^{(D)}$ w.r.t. $s_i^{(Q)}$. The objective of a neural model for text matching is to learn the optimal model Φ^* by minimizing a set of prediction errors. These errors are computed using a defined loss function $\pounds$, corresponding to every true label y_{ij} compared to the predicted value $\hat{y}_{ij}$. As described in equation 4.1.

$$\Phi^* = argmin \sum_i \sum_j \pounds \left(\Phi; s_i^{(Q)}, s_{ij}^{(D)}, \hat{y}_{ij}, y_{ij} \right) \tag{4.1}$$

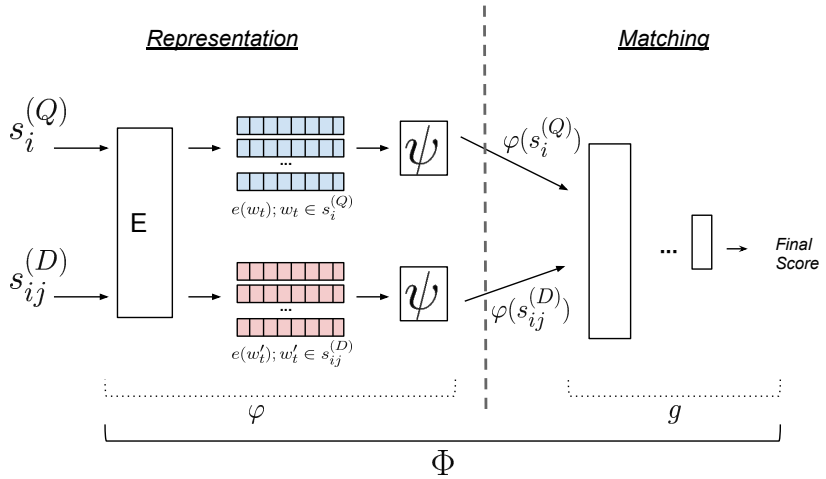


Figure 4.1: General framework describing a unified neural model for text matching.

where the estimated output label $\hat{y}_{ij}$ is computed by the matching model Φ that could be abstracted as in equation 4.2.

$$\Phi \stackrel{def}{=} g\left(\varphi(s_i^{(Q)}), \varphi(s_{ij}^{(D)})\right) = \hat{y}_{ij} \tag{4.2}$$

where φ is a representation function that extracts features from a raw text input, and that function ψ combines the embedded word representations $e\left(w_t\right)$ of an input sequence $s =< w_1 ... w_{|s|} >$.

$$\varphi(s) = \psi\left(e\left(w_1\right), ..., e\left(w_{|s|}\right)\right) \tag{4.3}$$

where $e : V \mapsto E$ is an embedding function that maps every word in a vocabulary V to a real valued structure in the embedding space E[4]. Hence, ψ is a representation function of input sequences, and that could be an aggregation-based (section 3.3.1) or not (section 3.3.2).

4.3.2 Representation-focused vs Interaction-focused

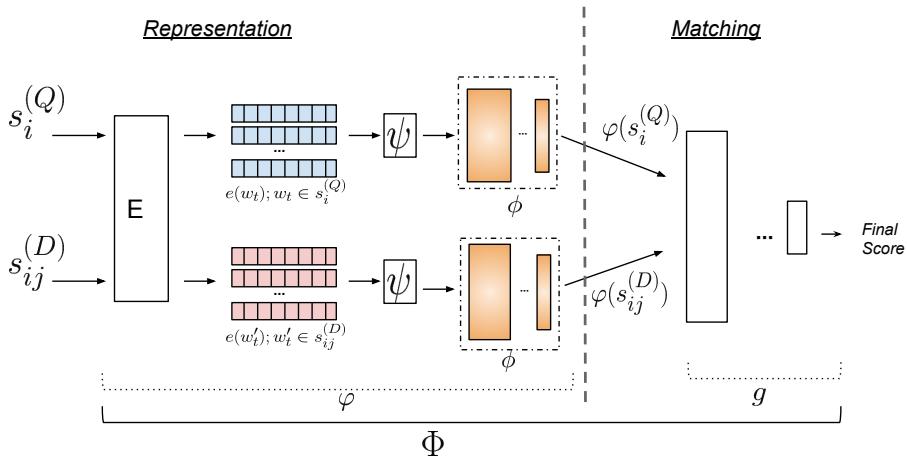
In this part, we describe and compare a set of representation-focused models [16, 237, 238, 239, 240, 241] and the interaction-focused models [178, 242, 19, 179, 17]. To describe the main differences between representation-focused and interaction-focused models, we consider two general architectures describing models of every class, as shown in figure 4.2.

In the followings, we highlight the different components of every framework.

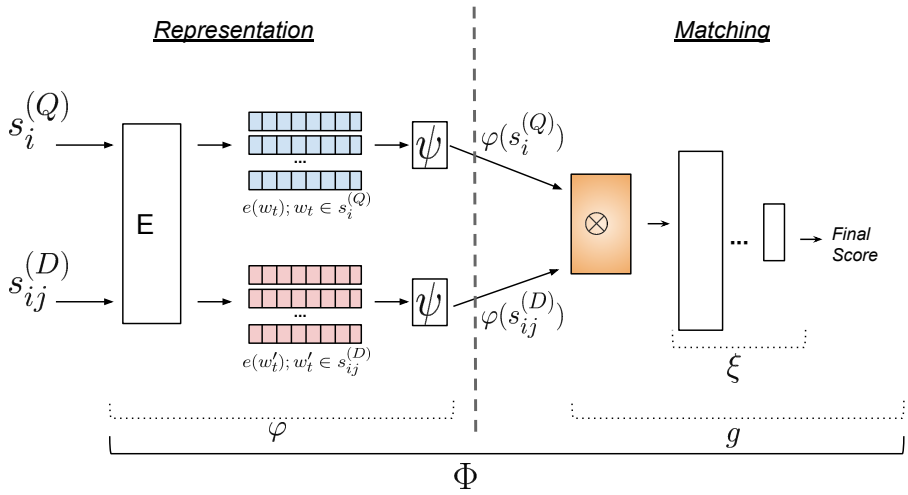
Representation-focused Models

Figure 4.2a illustrates a general framework for representation-focused models. Models of this class attempt to learn latent patterns that best represent the

[4]In case of embedded word vector representations, $E \subset \mathbb{R}^{dim}$ with dim is the dimension of the embedding space.



(a) The representation-focused framework



(b) The interaction-focused framework

Figure 4.2: A general architectures showing the main differences between the *interaction-focused* models and the *representation-focused* models.

input sequences. In the *representation* part, the function φ computes distributed representations $\varphi(s_i^{(Q)})$ and $\varphi(s_{ij}^{(D)})$ corresponding to the input sequences $s_i^{(Q)}$ (e.g. query or question) and $s_{ij}^{(D)}$ (candidate result, e.g. document or answer), respectively. As shown in equation 4.4.

$$\varphi(s) = \phi\left(\psi\left(e\left(w_1\right), ..., e\left(w_{|s|}\right)\right)\right) \tag{4.4}$$

where $e\left(w_t\right)$ is an embedded word vector of the sequences s. The representations $\phi(s_i^{(Q)})$ and $\phi(s_{ij}^{(D)})$ are then fed to the matching function g, and that can be a NN (e.g. CNN or RNN) or a simple matching function (e.g. cosine). ϕ combines the lower level features computed by the function ψ. This latter computes lower level (e.g. letter-level or word-level) features.

Several existing neural models for text matching could be put in this category. Huang et al [16] proposes a deep structured semantic model (DSSM) for ad-hoc web search. The DSSM network consists of two symmetric deep branches whose parameters are shared for both input sequences - specifically a query Q and a document D. All hidden layers of the DSSM model are stacked MLPs, used to compute intermediate semantic representations. Based on the general representation-focused model of figure 4.2a, in the DSSM model, the embedding space E contains a set of representation vectors learned in the letter-level. Called [16] *word hashing*. This model aims to reduce the dimensionality of the initial BoW term vectors that use all the dataset vocabulary. The intermediate representation function ψ corresponds to a concatenation of the different letter n-gram vectors of an input sequence (document and query). The representation function ϕ corresponds to the set of stacked MLP layers used to compute the semantic representations of input sequences [16]. While the matching layer g corresponds to the cosine simimarity function.

The work of Shen et al [237] extends the DSSM model by introducing a convolution neural network (CNN) into the DSSM architecture. In this model, called Convolutional Latent Semantic Model (CLSM) or convolutional DSSM (C-DSSM), instead of having a simple MLP layer in the representation function ϕ, a convolution layer is used to map the sparse initial representations computed by ψ into a semantic-based representations. The convolution layer transforms the tri-grams of the different words in the input sequences, into a latent vector then a max-pooling layer is used to extract the most significant local characteristics, and construct fixed length global vectors for the input sequences.
In other models, the representation function ϕ (figure 4.2a) could be a CNN of multiple convolution and pooling layers, as in the convolutional LTR model of Severyn and Moschitti [238] and the ARC-I model architecture of Hu et al. [17]. For instance, the convolutional LTR model [238] first, the function ψ concatenates the different word vectors then the function ϕ computes the embedded representation corresponding to every input sequence using a CNN layer. In the *matching* part, a first similarity signal called x_{sim} is computed using a learnable matching matrix M. This signal is aggregated, in a same features vector $X = \left[\varphi(s_i^{(Q)})||x_{sim}||\varphi(s_{ij}^{(D)}||x_{tfidf}\right]$ with the computed representations, where x_{tfidf} are *tf-idf* features relating the input sequences, and $||$ refers to a concatenation. The resulting vector X is then fed to the matching function g which corresponds to an MLP layer, to build an LTR-like classifier.

In other cases, the representation function ϕ (figure 4.2a) is made of a recurrent NN. For instance, Sutskever et al. [78] proposed an end-to-end model that

learns to predict a target sentence supposed to be similar to a given input sentence, represented by its component word embeddings, using an LSTM encoder-decoder model. The MV-LSTM model of Wan et al. [3] uses a bi-LSTM layers in the function φ to learn position-based representations for every input sequence. The Skip-thought model [12] uses a GRU[5] layers to learn the sequence-to-vector mapping function φ, by predicting forward and backward sentences of a given current sentence. Some recent recurrent-based models [243, 241] combine representation signals computed by the hidden states of the recurrent networks of function φ, with external signals to strengthen the final representations. For example, in the RNN-*Similarity* model [243], Kamath et al. proposed to integrate sequence-level features, such as expected answer types (e.g. location, human, entity or number) in the field of sequence answer selection. These features are based on the semantic information of the input question, and used in order to learn the matching as well as the representation functions of the final model. Differently from RNN-*Similarity* model[243], in the DRCN model proposed by Kim et al. [241], each layer of the representation learning branches (figure 4.2a) provides a concatenated information of attention-based features as well as hidden features of all the preceding recurrent layers, in order to enable preserving the original and the attention-based information about the input sequences. All the information, from the bottommost word embedding layer to the uppermost recurrent layer, are concatenated in the same features vector and fed to a fully connected layer to compute the final matching score.
Furthermore, both of the recurrent and a convolution layers could be combined [240, 244] in the same representation function ϕ, for capturing the context dependencies with convolutional layers, taking into account the sequencing of information by the recurrent layers.

Interaction-focused Models

In the interaction-focused framework, the model attempts to learn different matching features given the initial distributed representations of the input sequences being matched. The objective in this framework is to make the inputs interact earlier in the model [178], in order to extract matching signals in a lower level compared to the representation-focused models. The general framework that can illustrate the architecture of such models is presented in figure 4.2b, where *representation* part computes initial representations of the input sequences, by combining their component word vectors in the function ψ. In the *matching* part, the layer $\otimes$ refers to the function that extracts interaction features from the word-level, and the function ξ refers to the subsequent layers that compute relevance features of a higher levels. Hence, the matching function g can be redefined as in equation 4.5.

$$g\left(\varphi(s_i^{(Q)}), \varphi(s_{ij}^{(D)})\right) = \xi\left(M\right) \tag{4.5}$$

where M is a word-level matching matrix computed as in equation 4.6, and which varies depending on the different models of the state-of-the-art. ξ is an interaction function applied to the word-level signals, as highlighted in figure

[5]Is a specific LSTM architecture that, differently from the basic LSTM, GRU uses a gating mechanism to track the state of a given sequences without using separate memory cells [31]

4.2b.

$$M = \otimes \left(\varphi(s_i^{(Q)}), \varphi(s_{ij}^{(D)}) \right) \tag{4.6}$$

where $\otimes$ is a matching function (e.g. cosine).

Three different types of NN architectures could be used to learn the interaction function ξ of figure 4.2b. In case a CNN model is used [17, 19], the objective is to learn a contextual features. Hence, a sliding window $c^{Q \times D}$ is defined and moved in the matrix M, corresponding to a combination of two sliding windows, $c^{(Q)}$ and $c^{(D)}$, considered in the input sequences, $s_i^{(Q)}$ and $s_{ij}^{(D)}$ respectively, at the same time. In case an MLP architecture is used [178, 245], the objective is to learn different interactions between the matching features from the word-level. Finally, in case a recurrent architecture RNN is used [27, 246], the objective is to learn sequence and structure-based features. Hence, the matrix M is considered as an ordered sequence of information.
Regardless the function ξ, the input texts can be represented in different ways, including classical *one-hot* word vectors [179], pre-trained embedded representations [17], or representations that are updated (learned) during the model's training [19].

One of the models that use a CNN architecture to learn the function ξ is the ARC-II model proposed by Hu et al. [17]. This model uses an interaction matrix M computed using a product scheme between input query and document matrix representations. These representations are constructed by accumulating the component sequence word vectors. The intuition is to capture the interactions between words in a latent space rather than the word-word co-occurrences in the whole corpus. This intuition is supported by the effective performance of ARC-II [17] on different text matching tasks, such as sentence completion, question answering and paraphrase identification. In [179], Mitra et al. emphasize the importance of lexical matching in deep neural models for IR. They showed that representation-focused models tend to perform less when faced with rare terms. They also argued that web search requires both exact and non-exact (semantic-based) matching. Indeed, this type of search involves the use of several rare terms in some queries [179], and these are not necessarily used in the training collections of the latent representation spaces, and the exact matching will be more accurate to find the appropriate documents. Based on this motivation, the authors proposed a neural IR model, called DUET architecture, that incorporates both lexical and semantic matches. The DUET model consists of two parallel parts: the *local* model for exact matching that seeks to learn interaction signals between two texts; and the *distributed* model for semantic matching, based on the latent representation of texts. Final matching scores are computed by aggregating scores given by both the *local* and *distributed* parts of the DUET model [179].

Some other models define the interaction function ξ using an MLP network. For example, the DeepMatch model proposed by Lu and Li [242] applies a deep MLP to the interaction matrix M corresponding to a word-word co-occurrence matrix. This model considers a hierarchical decisions for matching at different levels of abstraction. Local decisions, which capture the interaction between semantically related words, are combined through the different hierarchical layers of the NN, to learn the global matching decision.
The DeepMatch model is trained [242] using triplet sequences $\left(s_i^{(Q)}, s_{ij}^{(D)+}, s_{ij}^{(D)-}\right)$,

where $s_{ij}^{(D)+}, s_{ij}^{(D)-} \in T_i$, and $s_i^{(Q)}$ is more similar to $s_{ij}^{(D)+}$ than to $s_{ij}^{(D)-}$. The objective is to maximize the similarity values computed by g (equation 4.5) for the positive pairs versus the negative ones, as described in the hinge-loss function of equation 4.7.

$$\pounds\left(s_i^{(Q)}, s_{ij}^{(D)+}, s_{ij}^{(D)-}\right) = \max\left(0, \epsilon + g\left(\varphi\left(s_i^{(Q)}\right), \varphi\left(s_{ij}^{(D)-}\right)\right) - g\left(\varphi\left(s_i^{(Q)}\right), \varphi\left(s_{ij}^{(D)+}\right)\right)\right) \quad (4.7)$$

where $\epsilon = 0.1$ is a parameter used to control the training margin.
In [178], Guo et al. proposed a deep relevance matching model (DRMM). They show that the deep learning methods built for semantic matching, would not be well suited to ad-hoc research. The latter concerns basically the relevance matching rather than the semantic matching. Based on this main difference, DRMM model computes the word-word interactions between input sequences, using the matching function $\otimes$ (equation 4.6) corresponding to a cosine similarity, then the word-level matrix M is transformed into histograms. Every matching histogram regroups local interactions according to their signal strength levels rather than their position.

In other cases, the matching function g (equation 4.5) uses mainly a recurrent NN. For example, in the work of Wan et al. [246], the authors propose the Match-SRNN model for position-based matching between input sequences. In the Match-SRNN model, given the input sequences $s_i^{(Q)}$ and $s_{ij}^{(D)}$, a word-level matching tensor M is computed using the $\otimes$ function defined in equation 4.6. Where every element m_{tk} in M corresponds to the interaction at position t of $s_i^{(Q)}$ and at position k of $s_{ij}^{(D)}$. This interaction is computed based in a combination of the prefix and the current word of every sequence. The matching matrix M is then fed to a the interaction function g that corresponds to a spatial RNN followed by an MLP layer to compute a final matching score.
Another RNN-based architecture is proposed by Fan et al. [27]. The authors highlighted that the main limitations of the document-wide models is that they perform a competition between long and short documents; while the passage-based models leverage simplified aggregation strategies of local signals but cannot well capture complex relevance patterns. To overcome these limitations, the authors proposed a Hierarchical Neural Matching model (HiNT) which consists of two stacked components [27]. Specifically, the *local matching layer* that uses an RNN architecture to compute a passage-level matching patterns; and the *global decision layer* that uses a second RNN architecture that performs interactions between the different passage-level signals to compute the document-level relevance features. The objective of the HiNT model is to assess, automatically, the relevance of long documents in the right information granularity, the word-level, the passage-level or the whole document. The local layer uses a spatial GRU [246] applied to a query-passage matching matrices. Then the global decision layer uses a hybrid network architecture to select signals from both passage and document level and compute the final matching score.

4.3.3 Attention-based vs Position-based

In some particular cases, specific assumptions about the input elements could be considered in the representation learning functions. Specifically, the *attention*

aspect that assumes that some elements are likely to be more important than others in a given sequence; and the *position* of these elements in that sequence. Hence, we distinguish the *attention-based* models and the *position-based* models, among others that we refer to as *general-features-based* models.

Attention-based Models

The *"attention"* concept, resulting from machine translation [31], has brought a significant gain in several NLP applications, including sentiment classification [25, 26] and paraphrase identification [236]. Attention-based models, identify the core information to be considered in a given sequence and allow focusing on some discriminated elements. The main idea is as follows:
Given an input sequence $s =< w_1...w_{|s|} >$, the attention model is expected to learn a coefficient vector α that determines how much attention should be given for each element in w_t. The elements in s will be then weighted accordingly. Hence, the attention vector $\alpha = [\alpha_1, ..., \alpha_{|s|}]$ is used in order to perform an optimal weighting process according to the task to be performed. Where every weight value α_t represent the amount of attention to be given to the word at position t of the sentence. These values are computed [157] as described in equation 4.8.

$$\alpha_t = \frac{exp(\boldsymbol{V}^T.w_t)}{\sum_{t=1}^{|s|} exp(\boldsymbol{V}^T.w_t)} \tag{4.8}$$

where $\boldsymbol{V}$ is a model parameter and represents the attention coefficients vector for the input sentence, and $\boldsymbol{V}^T$ its transposed vector.

Several authors have proposed attention-based models for text matching. For instance, Zichao Yang et al [25] propose a hierarchical attention network for document classification. The model combines a word and a sentence attention levels in a recurrent model architecture. The input text is first represented using word embeddings, then a first bidirectional GRU (bi-GRU) layer is applied to compute a representation vector for the input text. This vector is then passed through an MLP layer in order to construct an intermediate representation vector that will be used to learn attention coefficients of the different words. A second bi-GRU layer is used to compute the sentence level representation and learn the corresponding attention coefficients. The same process as in the word level is repeated in the sentence level. The final score is then computed based on these representations. Liu Yang et al [157] propose an attention-based neural model (ANMM) for question-answer matching. The input question and answer sequences are represented by their embedded word vectors. Then an interaction matrix is computed using the cosine similarities. The ANMM uses a neural architecture with a value shared weighting scheme, and a gating function where the question words are used to compute attention scores for the final matching signals. The shared weighting scheme [157], called *bin-sum*, aims at using the same connection weights for some nodes of the fully connected layer, where signals are in the same interval that represents the matching signal strength.
Recently, Peng and Liu [22] proposed an attention-based convolutional neural model for short text matching. The model is made of two modules that work in parallel: the word-level and phrase-level. The model combines the word overlap feature of the input text sequences, represented as word vectors, with the features learned by the NN, to compute the relevance probability of the

answer. First, matching matrices are computed for both the word-level and the phrase-level, using the cosine function, then a pooling layer is applied to both the matching matrices, in order to select the most important matching signals from the word level and the phrase level in parallel. Then, attention weights are computed at word and phrase levels, using the ReLU activation function for the input text representation. Finally, the word-level and the phrase-level score vectors are concatenated and fed to a hidden fully connected layer, to compute the final output probability.

Position-based Models

Differently from the attention-based approach, in the position-based models [3, 247, 24] the matching function is provided with the positional information corresponding to the different words in the input sequences being compared.

The word position is already considered, implicitly, in convolutional and recurrent models for text matching. In convolutional models [185, 17], the convolution layers process selected windows or n-grams of the input text in order to extract contextual features. In recurrent models [12, 78, 239], the words ordering of a given input sentence is processed in the different gates of the a RNN node (section 2.3.2). However, only the hidden activation vector corresponding to the last input word, is considered as the embedding vector of the whole sentence. Position-based models [247, 3] consider the position of every individual word in a given text, as an important factor to determine its weight. In the model proposed by Hui et al. [247], a semantic matching matrix is first computed for the embedded word vectors of the document and the query. This matrix is then distilled, by selecting the k most significant matching signals along the document dimension. The objective is to localize the relevance matching over all the matrix entries. This matrix is then fed to the DRMM model [178] architecture as input, to compute the final matching score.
Wan et al. [3] proposed the MV-LSTM model for matching sentences using representations computed at different positions of every sentence. The model uses bi-LSTM networks (section 2.3.2), where every hidden state computes a different representation for the input sentences. Hence, each input sentence acquires a different representation at every position. For example, consider the input sentence $s =< w_0 w_1 ... w_{|s|} >$, where w_t is the word w at position t. The hidden states $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$, computed as defined in equation 2.5, are concatenated to construct the bidirectional representation vector $s_t = \left[\overrightarrow{h_t}, \overleftarrow{h_t}\right]$ at position t of the sentence s. Hence, for two input sentences $s_i^{(Q)}$ and $s_{ij}^{(D)}$, the MV-LSTM [3] computes interaction matrices based on the bidirectional representations of each sentence. These matrices are then fed to a pooling layer that extracts k strongest interaction signals. Called *k-Max pooling*, this process is used in several neural text matching models [248, 185, 238, 21]. It aims at removing the noisy features from the interaction tensor.

Recently, Song et al. [24] proposed the positional convolution neural matching model (P-CNN). This model considers positional influence and interaction in multiple levels for text matching. The P-CNN first encodes the signals of positional information at the word level, the phrase level, and the sentence level. Then, a position-to-similarity mapping layer is defined to transform word-level positional information into local matching signals. At the phrase-level, a CNN

layer with position-sensible filters is used. A final matching function is then used to aggregate positional information, from the word and phrase levels, and compute a final matching score.

4.4 Discussion

Several issues are related to the use of deep models in text matching applications. We identify two main factors, the data and the models. In the following, we are going to discuss some issues related to every factor.

The type of data that is required in text matching applications, such as question answering and ad-hoc retrieval, consists mainly of three parameters: (1) a set of generalized queries, (2) a set of generalized documents, (3) relevance judgments (explicit human decisions or implicit such as click data). These three elements are more than required in order to design deep models in text matching applications. It is also useful to distinguish between deep neural models that focus on ranking long documents, from those designed for short text matching (e.g., for the question answering task, or for document ranking where the document representation is based on a short text field like *title*). The challenges in both of them are different, and DNN models need to be designed accordingly [249]. In one hand, when computing similarity between pairs of short texts, the vocabulary mismatch problem is more likely to happen than when retrieving long documents which could contain thousands of words and query words are more likely to be used [250]. In another hand, long documents may contain mixture of many topics and the query matches may be spread over the whole document. The neural ranking model must effectively aggregate the relevant matches from different parts of a long document [251, 27].

Another issue is related to the adaptability of neural models for the different applications of text matching. Indeed, the structures of deep models designed for text matching are often tailored to perform on specific datasets [249]. In addition, IR tasks deal with text of variable length, from short sequences (in case of answer selection) to long documents. This issue makes it inappropriate to directly adapt a deep model, designed for short text matching application, to the ad-hoc document retrieval.

Finally, text matching applications are different according to the task being addressed. In recent studies [11, 252], a new classification of neural text matching sate-of-the-art is provided. The different neural models for text matching could be divided according to the way of handling the different inputs to perform the task being addressed. In a previous work [252], we have introduced the *asymmetry* aspect regarding the different text matching applications in IR, and how the input sequences could be processed in order to perform the target task. This aspect will be discussed in more details latter in chapter 7.

4.5 Conclusion

In this chapter, we reviewed several neural text matching models, and provided a road-map of using the deep neural networks in different text matching applications.

The trend to use deep models in text matching is motivated by several limits

of the classical LTR methods, mainly the features engineering which is costly and time consuming. Hence, the deep models designed for text leverage the computational ability of NNs to automatically compute and combine relevance signals. We defined a unified neural matching model that we used to describe different state-of-the-art models. All the different neural matching models described in this chapter handle different matching tasks independently of the nature of the data and the task itself. However, the analysis made in [249] show that models designed for short text matching are inappropriate for ranking long documents. Finally, we described the models based on positional features and those putting a specific attention on particular words in a text sequence. We noticed that the positional features enable to enhance state-of-the-art results, as well as attention models do. However, to the best of our knowledge, there is no model that has considered both positional features and attention weight in the same model.

This chapter closes the state-of-the-art part of this document. In the following part, we will detail our contributions made during the preparation of this thesis, to solve several problems and issues discussed in this part. In particular, concerning the exploitation of distributed representations in IR, as well as the use of deep models in text matching applications.

Part IV

Contributions

CHAPTER 5

EXPERIMENTAL SETUP

5.1 Introduction

In this chapter, we describe the experimental setup used to evaluate the different contributions and baselines. This setup includes the datasets, the evaluation metrics, the frameworks and technical tools used to perform the different experiments. We also describe the different baseline models that we considered to compare the performances of our models to the state-of-the-art results.

5.2 Datasets

In this work, we focus on two main tasks, short text matching and ad-hoc document ranking. Therefore, we used different datasets to run our experiments.

5.2.1 WikiQA

WikiQA is a set of question and sentence pairs, provided by Microsoft Research [4]. The questions have been collected from Bing query logs, and have an average length of more than 7 words. Each question is associated with a Wikipedia page that potentially contains the answer. The candidate answers have an average length of more than 25 words. They correspond to selected sentences from the *summary* section of the corresponding Wikipedia page that provides the most important information about every question's topic.

Table 5.1 gives some statistics about the WikiQA dataset. The original corpus contains 3047 questions and a total of 29258 sentences. It includes several questions that have neither correct nor wrong answers. These questions have been filtered in the version we use in our experiment and which is provided by MatchZoo[1]. Hence, the new WikiQA version that we used includes 2477 questions and a total of 24590 sentences. Such that for every question there is at least one correct answer and at least one wrong answer. A sample of the WikiQA dataset is given in table 5.2.

[1] https://github.com/NTMC-Community/MatchZoo/tree/1.0/data/WikiQA

Dataset	Original				MatchZoo			
Characteristic	Train	Valid	Test	Total	Train	Valid	Test	Total
#num questions	2118	296	633	3047	2118	122	237	2477
#num sentences	20360	2733	6165	29258	20939	1115	2536	24590

Table 5.1: Statistics of the original WikiQA dataset of Yang et al. [4], compared to the one processed in MatchZoo [5]

QuestionID	Question	DocumentID	DocumentTitle	SentenceID	Sentence	Label
Q16	how much is 1 tablespoon of water?	D16	Tablespoon	D16-0	This tablespoon has a capacity of about 15 mL.	1
Q16	how much is 1 tablespoon of water?	D16	Tablespoon	D16-6	It is abbreviated as T, tb, tbs, tbsp, tblsp, or tblspn.	0
Q23	how old is zsa zsa gabor's daughter?	D23	Zsa Zsa Gabor	D23-1	Gabor was also a socialite.	0
Q23	how old is zsa zsa gabor's daughter?	D23	Zsa Zsa Gabor	D23-5	She later acted in We're Not Married!	0

Table 5.2: Sample from the WikiQA dataset of questions and their corresponding answers and labels.

5.2.2 QuoraQP

The QuoraQP[2] dataset consists of over 404K question pairs which are either similar or not. Statistics of this dataset are provided in table 5.3. A data sample of the QuoraQP dataset is listed in table 5.4, where each question pair is given a different *id*, and it is composed of two different sequences (questions) with their corresponding identifiers *qid1* and *qid2*. The last column tells whether the question pair is similar or not (1 or 0 respectively).

Question pairs	Positive (duplicates)	Negative (non-duplicates)
404351	149306	255045

Table 5.3: Description of the experimental QuoraQP dataset.

[2]https://data.quora.com/First-Quora-Dataset-Release-QuestionPairs

id	qid1	qid2	question1	question2	is_duplicate
0	1	2	What is the step by step guide to invest in share market in india?	What is the step by step guide to invest in share market?	0
1	3	4	What is the story of Kohinoor (Koh-i-Noor) Diamond?	What would happen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?	0
5	11	12	Astrology: I am a Capricorn Sun Cap moon and cap rising...what does that say about me?	I'm a triple Capricorn (Sun, Moon and ascendant in Capricorn) What does this say about me?	1
7	15	16	How can I be a good geologist?	What should I do to be a great geologist?	1

Table 5.4: Sample of some question pairs from the QuoraQP dataset.

5.2.3 Ad-hoc Document Ranking Datasets

We used two different types of the Text REtrieval Conference (TREC)[3] collections, news and Web datasets. All these datasets have the same format specified by the TREC campaign. For each dataset, a set of queries and relevance judgments (Qrels) are provided. Statistics about all the TREC datasets are provided in table 5.5.

Collection	# of documents	TREC query ids
AP 88-89	165 K	51 - 200
Robust04	528 K	301 - 450 601 - 700
GOV2	25 M	701 - 800

Table 5.5: Statistics of the TREC datasets used for ad-hoc document ranking.

News Datasets

We used two different datasets:

- *Associated Press (AP).* This is a set of news papers containing over 165K documents of different years (1988-1990). In our experiments, we only used documents of the AP-1988 and AP-1989, which correspond to the common use in IR literature, and for which relevance judgments are provided for 150 different queries.
- *Robust04.* This contains documents of the TREC 2004 Robust Track[4]. The dataset consists of news papers documents in discs 4 and 5 of the TREC data excluding documents of the Congressional Record set. This dataset contains over 528K documents and 250 judged queries.

[3]https://trec.nist.gov/data.html
[4]https://trec.nist.gov/data/t13_robust.html

Web Dataset

We used one Web dataset which is the GOV2 dataset. This dataset is used in the TREC Terabyte Track[5]. It contains a large proportion of the crawlable pages from .gov, in both HTML and text formats. Text documents may come from original PDF, Word or Postscript files. The GOV2 collection is 426GB in size and contains over 25 million documents. The GOV2 dataset includes 100 queries and the corresponding relevance judgments for the different associated documents.

5.3 Evaluation metrics

We have used different metrics in order to evaluate the performance of our proposed models as well as the different baselines. Several evaluation measures have been previously defined in section 1.5.1. In this section, we cite the different measures that we used in our experimental process, and give a brief reminder of their definitions and the reasons for our choice.

- *MAP.* Our aim is to evaluate for each query the distribution of relevant documents in the set of returned results.
- *nDCG@k.* We used this measure at different ranks, 1, 3, and 5 in the evaluation with a question-answering task; and at 5, 10 and 20 while evaluating with the ad-hoc document ranking task.
- *P@k.* The precision at different ranks is used to promote the relevant documents of a ranked results list. We evaluated the precision at different ranks, 1, 3, 5, 10, and 20.
- *MRR.* We used this measure for evaluations in the question-answering task. We assume that the user is likely to be satisfied if the correct answer is found in the top of results list.
- *Acc.* In our experiments, we only used this measure while evaluating the neural models in the paraphrase identification task. We evaluated the models ability to correctly find all the positive question-pairs.

5.4 Baseline models

While evaluating our different contributions, we considered several baseline models, namely classical and neural models.

5.4.1 Classical models

We considered two classical models that have already proven their effectiveness in different text matching applications. In the following, we give a brief description of the different models. We provided more details part III.

- *BM25.* is a probabilistic weighting model defined by Robertson et al. [253].
- *LM.* we used the language model of Metzler [49].

[5] https://www-nlpir.nist.gov/projects/terabyte/

5.4.2 Classical models with word embeddings

We compared our models to some models where the word embedded representations have been used to enhance the classical ones.

- *RM-Cent.* [191] is a query expansion model using word embedded representations.
- *NWT.* [14] is a model where all terms of the query are compared indifferently to all terms of the document, using cosine similarities with learned weights.

5.4.3 Neural models

We have also considered a set of neural models from the MatchZoo[6] framework.

- *ARC-I, ARC-II.* [17], are two convolutional models for sentence matching. ARC-I is a representation-focused model and ARC-II is an interaction-focused model.
- *DSSM.* [16], is a model made of two symmetric deep MLP structures to compute semantic representations for the input sequences.
- *CDSSM* [185], is an extension of the DSSM model [16] with a convolutional layer.
- *DRMM* [178], a histogram-based model for ad-hoc retrieval.
- *DUET* [179], a convolutional model composed of two parallel modules running together, the *local* model and the *distributed* model.
- *MatchPyramid* [19], a hierarchical convolution-based model for short text matching.
- *MV-LSTM* [3], a position-based recurrent model for sequence representation and matching.
- *ANMM* [157], is an attention-based neural model for short text matching.
- *KNRM* [180], is an end-to-end neural matching model for ad-hoc retrieval.

5.5 Tools and frameworks

We used different tools to perform the experimental and evaluation processes. Here, we give a brief description of those tools.

- *MatchZoo.* [5] is a framework for implementing, experimenting and comparing neural-based text matching models. This framework is developed with Python using different deep learning libraries such as Keras[7] and TensorFlow[8], in addition to other libraries that are developed mainly for

[6] https://github.com/NTMC-Community/MatchZoo/tree/1.0/matchzoo/models
[7] https://keras.io/
[8] https://www.tensorflow.org/

the different text matching processing steps, such as the matching tensors computation. Two main versions of the MatchZoo framework are provided. Specifically, the MatchZoo.1.0[9] release which is a software-like module, where models could be run using a command line; and the MatchZoo.2.x[10] which is still under development and is a library-like package. In our experiments, we only used the first one (MatchZoo.1.0) which is a first stable release, on which we have run several models, modified some of them and added new ones.

- *INDRI*. [254] is a text search engine developed as a part of the Lemur Project[11]. This tool, designed for academic purpose, and is used to preprocess, parse and index different text datasets. INDRI can parse TREC newswire datasets and web collections. It allows running a set of baselines and returning the results in the TREC standard format. We used INDRI to parse and index the different datasets that we used in the experiments, as well as to run different classical baseline models, such as BM25 and LM.

- *Pyndri*. [255] is a python interface to the INDRI search engine. It is designed as an integrated Python library dedicated to IR research. Pyndri[12] offers read-only access at two levels in a given INDRI index, the dictionary and tokenized document collection, as well as the queries evaluation on the constructed index. In our experiments, we first indexed the different datasets using INDRI, in order to have an homogeneous data pre-processing (tokenization, lemmatization ...), then used Pyndri for accessing the different indexes.

- *Trec_eval*[13]. [256] is the standard tool used by the TREC community for evaluating a ad-hoc retrieval models. Given the results file (run) produced by the model being evaluated and a standard set of judged results, this package provide a list of performance values in terms of different metrics, such as MAP, nDCG and precision.

5.6 Conclusion

In this chapter, we described all the experimental set up that we adopted during the preparation of this thesis. Mainly, we described the different datasets, evaluation measures, baseline models and finally the technical tools. Instead of using MatchZoo to run the different neural baselines, one could use the original implementation corresponding to every model. In our case, we used MatchZoo since the different baselines are implemented using the same libraries and use datasets of a same format.

[9] https://github.com/NTMC-Community/MatchZoo/tree/1.0
[10] Accessible in Aug 2019: https://github.com/NTMC-Community/MatchZoo/tree/2.2-dev
[11] The Lemur Project develops search engines, and other tools for text analysis to support research and development of IR and text mining. It is accessible in: https://sourceforge.net/projects/lemur/
[12] https://github.com/cvangysel/pyndri
[13] https://github.com/usnistgov/trec_eval

CHAPTER 6

QUERY WORDS IMPACT IN DOCUMENT RANKING USING WORD EMBEDDINGS

6.1 Introduction

In the previous chapters 1 and 3, we discussed several limitations of the models using the classical BoW representations, mainly the vocabulary mismatch. To solve this problem, recent approaches [257, 191, 154, 173] rely on the use of distributed representations of words [55, 9], enabling a semantic-based matching.

Most of the models described in section 3.4 compare all query terms with all document terms in the same way. These models do not distinguish between query terms that are present in the document and those that are not. We believe that the absence of a query word in a relevant document is an important factor in the matching process, because query terms that are absent in the relevant document can appear in another semantic form in that document.

In this chapter, we analyze the impact of query terms that are not present in the relevant documents, on the document-query matching using embedded word vectors. To do so, we experiment different document-query matching strategies. We combine the exact matching of classical IR models with the semantic-based matching of distributed word representations. We studied three different matching strategies:

- Comparison of all query terms with all document terms in the same way.
- Comparison of the terms of the query with those of the document, according to their presence/absence in that document.
- Discriminatory comparison of the query terms with the document terms, based on their presence in that document and their similarities based on exact matching.

The chapter is organized as follows: in section 6.2, we give an example of a query and two candidate documents, where the relevant document does not contain as much query words as the irrelevant one. In section 6.3, we describe the classical query-document matching, that is then extended, in section 6.4,

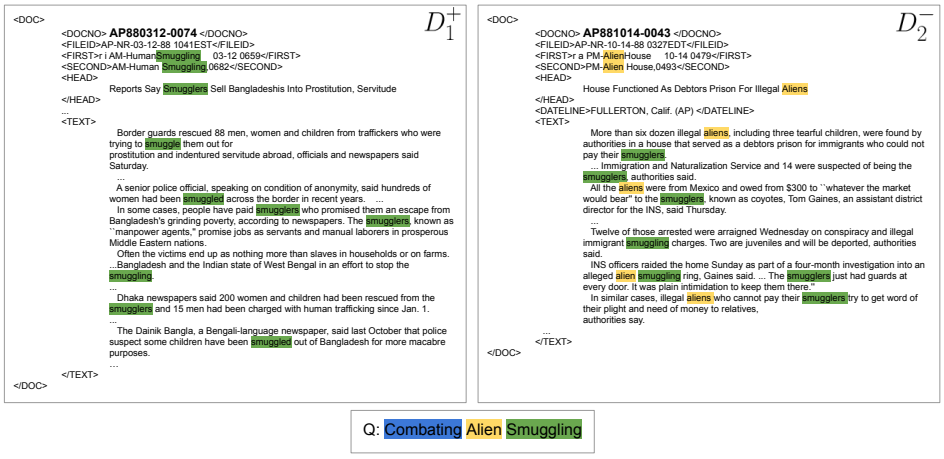


Figure 6.1: Highlighting query words occurrences in a relevant document D_1^+ and an irrelevant document D_2^-.

to propose different matching strategies using word embeddings. In section 6.5, we describe the experimental setup and the evaluation results.

6.2 Motivation

In most IR models using word embeddings, the absence of query words in the relevant documents is often addressed by mean of the query expansion [191, 257, 192], but the words in the new query are processed in the same way. Other models [14, 159, 153] compute the query-document similarity score by using all interactions between the words in the document and those in the query. The presence/absence of query words in a document is not explicitly considered in these models. Figure 6.1 shows a concrete example where a relevant document has several missing query words compared to an irrelevant document, where more query words have been used. Let us consider the query $Q =$"*Combating Alien Smuggling*", for which the document *AP880312-0074* is relevant, and we refer to it as D_1^+; and the document *AP881014-0043* is irrelevant, and we refer to it as D_2^-. Both D_1^+ and D_2^- are taken from the *AP* TREC dataset. Using the BM25 model implemented in INDRI, the system ranked the document D_1^+ at position 179 while the irrelevant document D_2^- is ranked at first position of the list. In figure 6.1, we show some snippets of documents, D_1^+ and D_2^-. We used three different colors to highlight the query words and the occurrences of their stems in each document. We can notice that, the relevant document D_1^+ contains only one query word for which the stem is *"smuggl"*. While the irrelevant one D_2^- contains more query words, of stems *"smuggl"* and *"alien"*. This example highlights the fact that the occurrence of all query terms in a document does not imply that this document is relevant.

6.3 Classical Query-Document Matching

In *tfidf* models based on the standard BoW representation, the relevance score sc assigned to a document D w.r.t. a query Q is generally computed [258] based on the exact match between every query term q_i and document term d_j, as shown in the general equation 6.1.

$$sc(Q, D) = \sum_{q_i \in Q \cap D} M_{tfidf}(q_i, D) \tag{6.1}$$

$M_{tfidf}(q_i, D)$ is the weight corresponding to the query word q_i in the document D, and is computed using a classical *tfidf* or any IR model.

This approach assumes that the query terms contribute to score sc only if they are present in the document, while those that are not, even if they contribute to the description of the information sought by the user in the query. These terms are not considered in this approach, which results in some information missing. To cope with this limitation, distributed representations of words [2, 9] can be used. These representations enable to exploit the semantic similarities between two words lexically different w_i and w_j, such that the semantic relatedness is translated by a function sim computing a distance between the corresponding vectors $\vec{w}_i$ and $\vec{w}_j$.

In the following sections, we describe different matching strategies, described in an earlier work [259], and allowing to exploit the semantic links between the vectors of the query words and those of the document, based on the occurrence of the query words in the documents.

6.4 Matching Strategies Using Semantic Word Similarities

One simple way is to compare all query terms with all the terms of the document using their representation vectors, as in [14, 260]. Thus, the importance of every term in the document is assessed w.r.t. the terms of the query, as described in equation 6.2:

$$sc(Q, D) = \sum_{d_j \in D} \sum_{q_i \in Q} M_{tfidf}(d_j, D) \times sim(q_i, d_j)^{\alpha} \tag{6.2}$$

$M_{tfidf}(d_j, D)$ is the weight of the document term d_j, $sim(q_i, d_j)$ is a normalized semantic similarity between the terms q_i and d_j term. α is a parameter used to control the impact of the semantic similarity $sim(q_i, d_j)$.

Based on the comparison of all query words with all document words in equation 6.2, we can transform this equation to better highlight the aspect of presence/absence of query terms in the document (equation 6.3) and/or on the exact or lexical matching between the terms (equation 6.4). In both these strategies, we consider a normalized word similarity function $sim(q_i, d_j)$.

6.4.1 Presence/Absence Split

In this strategy, we intend to observe the contribution of query terms that are not present in the document to the assessment of the document's relevance

score. We decompose the equation 6.2 into two parts: the first part deals with the terms of the query that are present in the document while the second part deals with the terms of the query that are not in the document. This process is shown in equation 6.3:

$$\begin{aligned} sc(Q,D) = \lambda \times \sum_{d_j \in D} \sum_{q_i \in Q \cap D} M_{tfidf}(d_j, D) \times sim(q_i, d_j)^{\alpha} + \\ (1-\lambda) \times \sum_{d_j \in D} \sum_{q_i \in Q \setminus D} M_{tfidf}(d_j, D) \times sim(q_i, d_j)^{\alpha} \end{aligned} \tag{6.3}$$

where λ is used to control the impact of each part in the equation.

6.4.2 Exact/Semantic Matching Split

We have decomposed the first part of the equation 6.3 into two components, as shown in equation 6.4. In this equation, we can separately observe the impact of the following elements:

- The exact matching of the query terms and the document terms, which is described by the first part of equation 6.4 ($\sum_{q_i \in Q \cap D} M_{tfidf}(q_i, D)$);
- The semantic matching between the query terms in the document and its other terms, which is described by the second part of equation 6.4 ($\sum_{q_i \in Q \cap D} \sum_{d_j \in D \setminus \{q_i\}} M_{tfidf}(d_j, D) \times sim(q_i, d_j)^{\alpha}$);
- The semantic similarity between terms of the query that are not in the document with the terms of the document, which is expressed in the third part of equation 6.4 ($\sum_{d_j \in D} \sum_{q_i \in Q \setminus D} M_{tfidf}(d_j, D) \times sim(q_i, d_j)^{\alpha}$).

.

$$\begin{aligned} sc(Q,D) = & \lambda_1 \times \sum_{q_i \in Q \cap D} M_{tfidf}(q_i, D) + \\ & \lambda_2 \times \sum_{q_i \in Q \cap D} \sum_{d_j \in D \setminus \{q_i\}} M_{tfidf}(d_j, D) \times sim(q_i, d_j)^{\alpha} + \\ & (1-\lambda_1-\lambda_2) \times \sum_{d_j \in D} \sum_{q_i \in Q \setminus D} M_{tfidf}(d_j, D) \times sim(q_i, d_j)^{\alpha} \end{aligned} \tag{6.4}$$

$\lambda_1 + \lambda_2 \leq 1$ are parameters used to control the impacts of the different items cited above.

6.4.3 Relations Between the Different Matching Strategies

In the previous matching strategies, note that if $\lambda = 0.5$ in equation 6.3 and if $\lambda_1 = \lambda_2 = 1/3$ in equation 6.4 then the equations 6.3 and 6.4 are equivalent to the equation 6.2.

Consider the matching function as defined in equation 6.5:

$$sim(w_i, w_j) = \begin{cases} 1 & if\ w_i = w_j \\ 0 & otherwise \end{cases} \tag{6.5}$$

If the similarity between the query terms and the document terms is computed by the equation 6.5, then the equation 6.1 will be a special case of each of the matching strategies described by the equations 6.2, 6.3, and 6.4. Indeed, when $\lambda = 0.5$ in the equation 6.3 and $\lambda_1 = \lambda_2 = 1/3$ in the equation 6.4, the equations 6.2, 6.3, and 6.4 become similar to 6.1.

6.5 Experiments

6.5.1 Evaluation methodology

Since the overall experimental framework of this work was described in chapter 5, we will briefly recall the experimental set up used in order to evaluate the different matching strategies and compare them. We used two traditional models to compute the M_{tfidf} function in each of the equations, 6.1, 6.2, 6.3 and 6.4, namely the BM25 [253] where following the common use in IR, parameters k_1 and b are set to 1.2 and 0.75 respectively; and the language model of Metzler et al. [49] where $\lambda_D = 0.2$ and $\lambda_C = 0.4$. We used the three TREC datasets described in table 5.5. The *AP 88-89* is used to set the parameters λ, λ_1, λ_2 and α of the different matching strategies defined in section 6.3. Performances are reported using *Robust04* and GOV2.

6.5.2 Parameter Setting and Impact Analysis

We use the cosine function to compute the similarity sim between every document term d_j and query term q_i. For the embedded word representations, we used the pre-trained Word2Vec[1] model, where each word is represented by a vector of 300 dimensions. Out of vocabulary terms are simply ignored. We conducted experiments on the dataset *AP 88-89* to set the different hyper-parameters. The parameter α takes values in $\{1, 2, ...20\}$ (above 20 the performances were unchanged), the parameters λ, λ_1 and λ_2 take values in $\{0.1, 0.2...0.9\}$. We used trec_eval to evaluate our model's performances in terms of MAP, $P@5$, $P@10$ and $P@20$ as well as the $nDCG@20$. The analyses of the equations 6.2 and 6.4 were similar to the one of the equation 6.3 for which we describe the impact of the different parameters that are used. Figure 6.2 shows the evolution of performance, in terms of MAP and $P@5$, using the BM25 model to compute M_{tfidf} in the equation 6.3. In this figure, each curve corresponds to a value of α.

We notice that MAP and $P@5$ gradually evolve in line with the increase in the value of λ for $\alpha \in \{1, 2\}$. For $\alpha \in \{3, 4, 5, 6\}$ the performances are stable for the values of $\lambda \in \{0.4, 0.5, 0.6\}$ then decrease slowly for $\lambda > 0.6$. For $\alpha = 7$ the values of MAP and $P@5$ become more stable with $\lambda \geq 0.4$.

This analysis shows the impact of the λ parameter used to analyze the influence of query terms that are not in the document on the matching process. In the equation 6.3, $\lambda = 0.4$ gives the best results, explaining the importance of query terms that are not in the document in the matching process. For $\alpha \in \{1, 2, 3, 4\}$ the performance is worse than for $\alpha = 7$, because the semantic similarity had more impact on the matching process. According to Zamani and Croft [154], since continuous representations of words are based on the notion

[1] https://code.google.com/archive/p/Word2Vec/

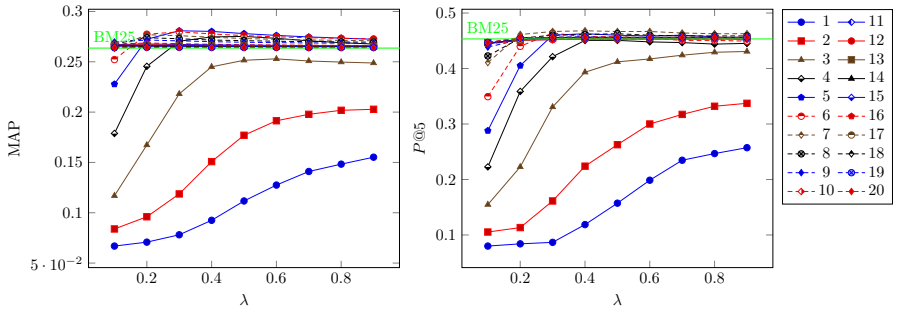


Figure 6.2: Analysis of the equation 6.3 sensitivity to parameters α and λ in the *AP 88-89* collection, in case of using BM25 as M_{tfidf}.

of context [2], they can lead to the use of irrelevant terms in document-query matching (e.g. the terms *secured* and *dangerous* have a great semantic similarity but *dangerous* is not relevant for the query *"security transport"*).

Based on the above analysis, we have defined the parameter values that correspond to the compromise between the performance in terms of $P@5$ and the MAP, leading to the following configuration: $\alpha = 7$ in both equations 6.2 and 6.3; and $\lambda = 0.4$ in the equation 6.3. For the equation 6.4, we use the following configuration: $\alpha = 5$, $\lambda_1 = 0.5$ and $\lambda_2 = 0.3$.

6.5.3 Results and discussion

As explained in section 6.3, we analyze the contributions of query terms, that are missing in the document, to the semantic-based matching process. In the following, we evaluate the different matching strategies to answer different research questions (RQ), where the labels *eq6.2*, *eq6.3*, and *eq6.4* correspond to the different matching strategies defined by the equations 6.2, 6.3, and 6.4, respectively.

RQ1: How parameter α impacts the performances of the three matching strategies?

We first analyze the behavior of each of the defined matching strategies, according to the different values of the α parameter. The figure 6.3 shows the evolution of the performances, in terms of MAP and $P@5$, of the different matching strategies.

In figure 6.3, according to the different curves, we can notice that all the equations behave in the same way with a slight difference in performance. The most important difference is that for $\alpha \in [4, 8]$, the performances of equations *eq6.3* and *eq6.4* are slightly better than the results of the equation *eq6.2*. However, for $\alpha > 8$ the performances of all the different matching equations decrease, because the contribution of the semantic similarity between words, $sim(q_i, d_j)$, is largely reduced due to the exponent value.
These results show, in one hand, that for the different equations 6.2, 6.3 and 6.4, the parameter α has the same impact; and in the other hand, the distinction between the query words that are in the document and those that are not (equations 6.3 and 6.4), has led to some improvement in the MAP and $P@5$

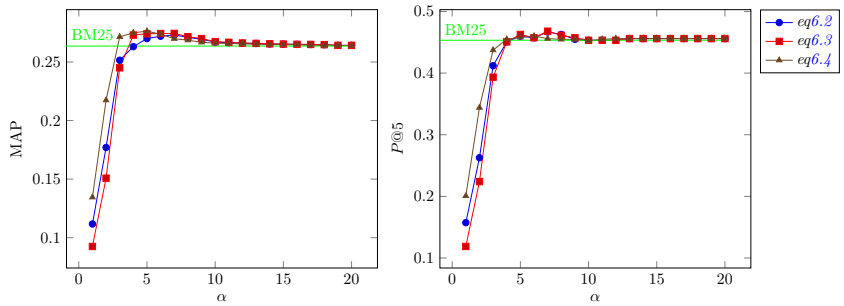


Figure 6.3: Comparison of the performances, in terms of MAP and $P@5$, of the different matching strategies, w.r.t. the different values of the parameter α in the collection *AP 88-89*.

values compared to the indifferent processing of all the query terms (equation 6.2).

RQ:2 What is the impact of a similarity threshold?

As suggested in several state-of-the-art analysis [14, 261, 192] that use the cosine similarity between word vectors, defining a threshold value to control the impact of this similarity is important. Hence, we evaluated the different matching strategies using a minimum threshold of the semantic similarity between words $\varepsilon \in \{0.0, 0.1, ..., 1.0\}$. The objective is that the similarities taken into account must be greater than the ε value. For this purpose, the similarity between a query word q_i and a document word d_j is computed as defined in the equation 6.6.

$$sim_\varepsilon(q_i, d_j) = \begin{cases} sim(q_i, d_j) & if \quad sim(q_i, d_j) > \varepsilon \\ 0 & otherwise \end{cases} \tag{6.6}$$

where $sim(q_i, d_j)$ is the cosine similarity between the representation vectors of q_i and d_j. Figure 6.4 shows the performances evolution of the different matching equations, according to the different values of ε, using the BM25 model in M_{tfidf} as defined in the section 6.3.

In this figure, we find that the ε threshold has a negative impact on the performance of each of the proposed matching strategies. We can see a similar behaviour of all strategies. For a low threshold value $\varepsilon \in [0.0, 0.2]$, the performance values gradually decrease as the values of ε become larger, $\varepsilon \in [0.3, 1.0]$. We used the α parameter to regularize the impact of the semantic similarity between words. Consequently, the small similarities (when $sim_\varepsilon(q_i, d_j) \approx 0$) are omitted, due to the property of the α power function near 0, and which have lead to a drop in performance, when an important values of ε is used.

RQ3: How the matching strategies are effective compared with classical models?

We also compared the different matching strategies that we propose with traditional IR models. Tables 6.1 and 6.2 show the results obtained on two datasets, Robust04 and GOV2, using BM25 and LM respectively, to compute the value of M_{tfidf} in each of the matching equations *eq6.2*, *eq6.3* and *eq6.4*. We reported

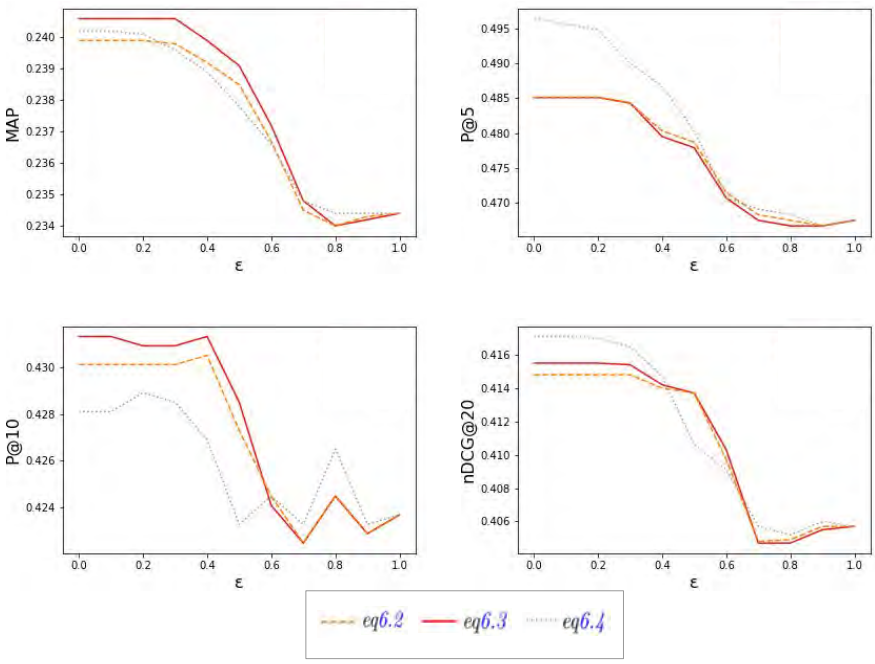


Figure 6.4: Evolution of the performances of the different matching equations with respect to the value of ε, in the Robust04 collection.

Dataset	Model	MAP W/L/T	P@5 W/L/T	P@10 W/L/T	P@20 W/L/T	nDCG@20 W/L/T
Robust04	BM25	0.2362	0.4747	0.4285	0.3528	0.4105
	eq6.2	**0.2400+** 152/85/13	**0.4851** 32/19/199	**0.4309** 37/32/180	**0.3548** 43/38/168	**0.4139** 120/86/43
	eq6.3	**0.2403+** 116/73/61	**0.4851** 1/3/246	**0.4301** 6/8/236	**0.3550** 15/12/223	**0.4140** 119/89/41
	eq6.4	**0.2401** 148/95/7	**0.4956+** 49/29/172	**0.4293** 52/53/145	**0.3772** 63/55/132	**0.4160** 125/96/29
GOV2	BM25	**0.2595**	**0.5463**	**0.5315**	**0.4977**	**0.4401**
	eq6.2	0.2554 90/56/4	0.5141 30/38/82	0.5040- 35/52/63	0.4893 46/42/62	0.4262 71/67/12
	eq6.3	0.2553 89/57/4	0.5154- 42/44/63	0.5013- 39/55/56	0.4893 53/53/44	0.4263- 75/65/10
	eq6.4	0.2524 86/60/4	0.5128 45/43/62	0.4933 45/54/50	0.4829 58/51/41	0.4161 73/69/8

Table 6.1: Experimental results using the BM25 in $M_{tfidf}(.,.)$ of the different matching strategies.

Dataset	Model	MAP W/L/T	P@5 W/L/T	P@10 W/L/T	P@20 W/L/T	nDCG@20 W/L/T
Robust04	LM	0.2310	0.4265	0.3936	0.3331	0.3807
	eq6.2	**0.2337** 162/75/13	**0.4378** 37/25/188	**0.4016** 40/32/178	**0.3392** 60/32/158	**0.3868** 29/37/183
	eq6.3	**0.2336** 119/81/50	**0.4345** 2/4/244	**0.4016** 8/9/233	**0.3392** 16/12/222	**0.3865** 29/37/183
	eq6.4	**0.2324** 166/77/7	0.4249 45/42/163	**0.3952** 59/52/139	**0.3438+** 82/50/118	**0.3856** 132/89/29
GOV2	LM	**0.2516**	0.4054	**0.4289**	**0.4094**	**0.3456**
	eq6.2	0.2446 81/65/4	**0.4201** 37/28/85	0.4161 43/46/60	0.4077 56/45/49	0.3392 79/57/14
	eq6.3	0.2444 81/64/5	**0.4201** 37/28/85	0.4161 43/46/61	0.4064 56/46/48	0.3384 79/58/13
	eq6.4	0.2277- 67/79/4	0.3544 43/52/55	0.3638- 45/55/50	0.3695 60/63/27	0.2954- 70/71/9

Table 6.2: Experimental results using the LM in $M_{tfidf}(.,.)$ of the different matching strategies.

the performances in terms of MAP, P@5, P@10, P@20 and $nDCG$@20 of each equation. The values in **bold** type represent the best performance over the different models. Results labelled with (+) or (−) show the significance [2] in, respectively, improvements or decrease in performance compared to the corresponding baseline model. W/L/T refer to the number of queries whose performance is improved (Win), deteriorated (Loss) or remained unchanged (Tied) by the corresponding matching strategy, w.r.t. the corresponding baseline.

In both tables 6.1 and 6.2, when considering the Robust 04 dataset, the different matching strategies perform better than the classical baselines, BM25 and LM. The improvements are significant, concerning MAP and P@5 with BM25 and in terms of P@20 with ML. The matching strategies *eq6.2* and *eq6.3* outperform the BM25 model with more than 2% in terms of P@5. The matching strategy *eq6.4* outperforms the BM25 model with more than 4% in terms of P@5. In the GOV2 dataset, both equations *eq6.2* and *eq6.3* outperform traditional models, BM25 and LM, in terms of only P@5. When the LM model is used to compute values of M_{tfidf}, the differences are not significant.

RQ4: How does the different query terms contribute to the relevance assessment?

In order to explain the results obtained in the different datasets, we analyzed the impact of the different query terms in the computation of the score of relevant documents in every dataset. Let $sc_t(D,Q)$ be the total relevance score computed for a document D according to the query Q defined as in equation 6.7. Note that in this analysis, we only consider documents that are judged as relevant.

$$sc_t(D,Q) = sc_a(D,Q) + sc_p(D,Q) \tag{6.7}$$

- $sc_a(D,Q) = \sum_{q_i \in Q,\ q_i \notin Q \cap D} \sum_{d_j \in D} sim(q_i, d_j)$ is the score calculated according to the terms of the query that are absent in the document;
- $sc_p(D,Q) = \sum_{q_i \in Q \cap D} \sum_{d_j \in D} sim(q_i, d_j)$ is the score calculated according to the terms of the query that are present in the document.

[2] We used the statistical test *t-test* with a confidence level of 95%.

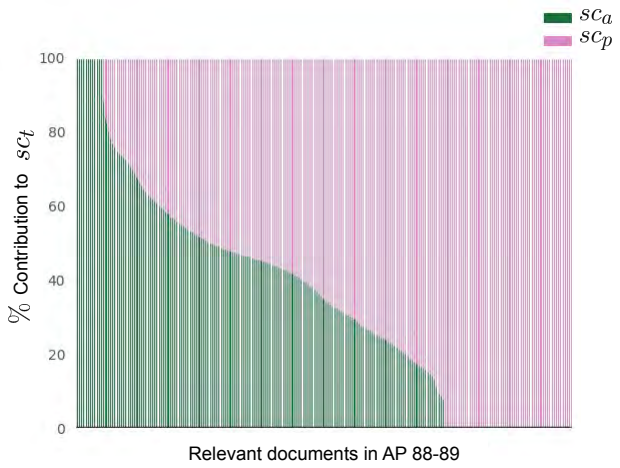


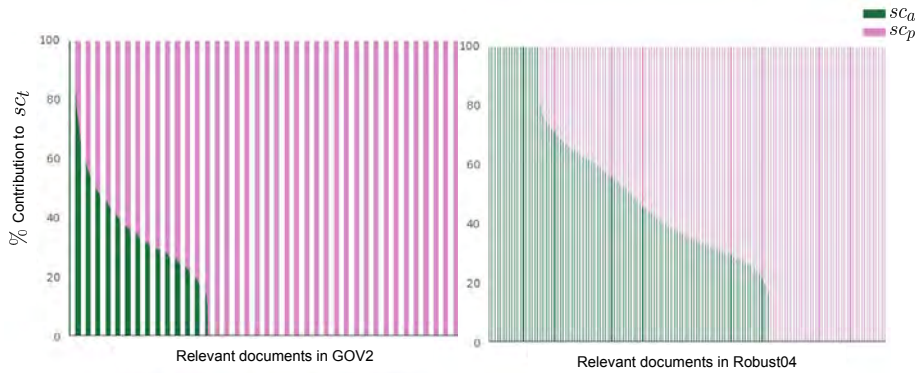
Figure 6.5: Contributions of different query terms, according to their presence/absence relevant documents, in the relevance score computation for relevant documents in the AP 88-89 dataset.

$sim(q_i, d_j)$ is the cosine similarity between the terms q_i and d_j.

First, we are going to analyze the proportion of relevant documents that are concerned with the problem of query words *absence* in the *AP 88-89* dataset used to set the different parameters of the studied matching strategies.

Figure 6.5 shows the percentages of contribution of the query terms that are present in the document, refereed to with sc_p, and the terms that are absent in the document, noted s_a, when computing the total score refereed to with sc_t for the relevant documents in the collection *AP 88-89*. Remember that we used this dataset to set the different parameters in each of the matching strategies we proposed. In this figure, each bar represents a relevant document whose score is computed by the equation 6.7. We can see that the scores of the majority of relevant documents (green area) are significantly affected by the similarities between the query terms that are absent in the documents and the terms of these documents.

Now, let us analyze the relevance scores of the test datasets, Robust04 and GOV2. Figure 6.6 shows the contribution percentages of the query terms that are present in the document, sc_p, and the terms that are not, sc_a, to the computation of the total score sc_t for the relevant documents in each of the GOV2 (figure 6.6a) and Robust04 (figure 6.6b) collections. In figure 6.6, we see that the scores of the relevant documents in the Robust04 dataset (figure 6.6b) are influenced by the similarities of the query terms that are not in the document (score sc_a) in a consistent way. This explains the improved research results obtained by the different matching strategies we proposed (tables 6.1 and 6.2). However, in the GOV2 dataset (figure 6.6a), the scores of relevant documents are not significantly influenced by the similarities between the query terms that are absent in the relevant documents and the terms of these documents (sc_a). In the Robust04 collection, nearly 12% of the relevant documents do not contain any query terms, so processing query terms that are missing in the relevant documents accordingly has worked well. Differently from the GOV2 collection



(a) (b)

Figure 6.6: Contribution of query terms according to their presence/absence in the relevant documents of the GOV2 and Robust04 datasets, to the total relevance score values.

that contains only 1.1% of relevant documents without any query term. This explains the non-influence of the split of the query terms according to their presence/absence in the documents of the GOV2 dataset.

***RQ5:* How the proposed matching strategies perform in comparison to state-of-the-art models?**

In this analysis, we evaluate how much the different matching strategies are effective compared to traditional model and some state-of-the-art models, where embedded word representations have been used to enhance traditional IR models.

In table 6.3, M_{BM25} and M_{LM} refer to the use of model BM25 and LM respectively to compute M_{tfidf} in equations *eq6.2*, *eq6.3* and *eq6.4*. *RM-Cent* corresponds to the model of Kuzi et al. [191] which is a query expansion model using the embedded representations of words. *NWT* refers to the model of [14], where all terms of the query are compared indifferently to all terms of the document. In this table, we report the results of the different approaches to the state of the art as presented in the corresponding articles for *RM-Cent* [191] and *NWT* [14]. The values in **bold** type represent the best performance over the different models

In table 6.3, we can notice that the matching strategies we studied outperform the state-of-the-art models in terms of P@5 in the Robust04 dataset and in terms of $nDCG$@20 in the GOV2 dataset. In all the matching strategies, we exploit all semantic interactions between the word vectors of the document and the word vectors of the query. Differently from models that process all query terms indiscriminately, RM-Cent [191] selects the most suitable terms for expanding the query, then the resulting query is used in a classic language model, to rank documents of the dataset; and the NWT model [14] where word vectors are used to capture semantic similarities between document terms and all query terms without distinction. Although the matching strategies that we propose are not more effective than all the baselines, the results support our intuitions

Collection	Model		MAP	P@5	P@10	P@20	nDCG@20
Robust04	M_{BM25}	*eq6.2*	0.2400	0.4851	**0.4309**	0.3548	0.4139
		eq6.3	0.2403	0.4851	0.4301	0.3550	0.4140
		eq6.4	0.2401	**0.4956**	0.4293	0.3772	0.4160
	M_{LM}	*eq6.2*	0.2337	0.4378	0.4016	0.3392	0.3868
		eq6.3	0.2336	0.4345	0.4016	0.3392	0.3865
		eq6.4	0.2324	0.4249	0.3952	0.3438	0.3856
	RM-Cent		**0.2910**	0.4950	-	-	-
	NWT		0.2740	-	-	**0.3800**	**0.4260**
GOV2	M_{BM25}	*eq6.2*	0.2554	0.5141	**0.5040**	0.4893	0.4262
		eq6.3	0.2553	0.5154	0.5013	0.4893	**0.4263**
		eq6.4	0.2524	0.5128	0.4933	0.4829	0.4161
	M_{LM}	*eq6.2*	0.2446	0.4201	0.4161	0.4077	0.3392
		eq6.3	0.2444	0.4201	0.4161	0.4064	0.3384
		eq6.4	0.2277	0.3544	0.3638	0.3695	0.2954
	RM-Cent		**0.3350**	**0.6230**	-	-	-
	NWT		0.304	-	-	**0.5240**	0.4220

Table 6.3: Comparison of the different matching strategies with some state-of-the-art models in two different datasets.

about the importance of query words that absent in relevant documents, in the relevance assessments.

6.6 Conclusion

In this chapter, we have analyzed the document-query matching process using word embedding in classical IR models.

We proposed different matching strategies based on the presence/absence of query terms in the document. We used semantic similarities between document and query terms to address the vocabulary mismatch between the document and the query. The results show that explicitly taking into account query terms that are not used in the document improves the matching process, and provide better results than traditional models based only on exact matching. The results are comparable to those of some state-of-the-art models in the Robust04 dataset. However, using the GOV2 collection, the different matching strategies do not provide a clear improvement, as the documents in this collection differ to the one in Robust04. In particular, documents in GOV2 are longer and they contain most of the query terms (which is shown by the analysis in the section 6.5.3).

The embedded representations of words are also used in neural matching models, in order to construct representations of whole sequences being matched. In the next chapter, we present our work on text matching with neural models.

CHAPTER 7

NEURAL MODELS FOR SHORT TEXT MATCHING USING ATTENTION-BASED MODELS

7.1 Introduction

We define the nature of a text sequence according to its length, that can be short or long, and its usefulness that can be providing or seeking for information. Our objective in this chapter is two folds: the first is to handle two different matching tasks according to their nature; the second is to study and evaluate the impact of some word-level features, such as position and attention, in matching different sequences. To do so, we first define the nature of the matching task, based on the type of the inputs and the relationship between them. We distinguish two types of matching, the *symmetric matching* and the *asymmetric matching*. In the *symmetric* tasks, such as document classification [19], we assume that inputs and outputs are interchangeable. In the *asymmetric* tasks, such as question-answer matching [185], the inputs are different and are not interchangeable. To handle these aspects, we propose a general architecture extending several state-of-the-art neural matching models, using attention layers. In a second step, we argue that the combination of different word-level features, such as attention and position, may lead to better results. In one hand, *Position-based* models [247, 3] rely on the word position to compute representations of the input text. The positional information of the word, such as proximity, word dependencies, and the sequence structure, are important in learning text representations and can have an important impact on the matching process. On the other hand, *Attention-based* representation models [26, 236, 157], learn coefficient vectors that put more attention on some words and enable to designate the most important words in the input text, regardless of their position. Hence, we believe that it would be interesting for a representation learning model to have information about the most important words, as well as their positions in the input text.

In this chapter, we present our neural approach for text matching. We will first motivate and detail the proposed models, in section 7.2. We then describe

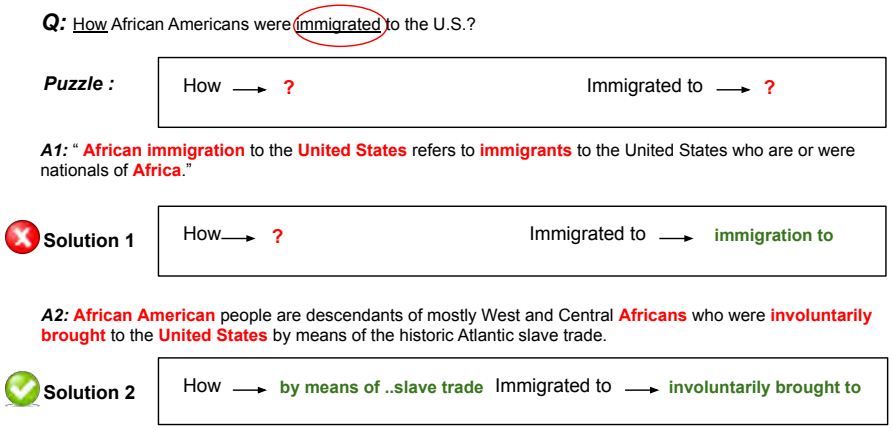


Figure 7.1: Solution 2 fills completely the missing part described in Q.

the experimental process and the evaluation results, in section 7.4.

7.2 An asymmetry sensitive approach for neural texts matching

Most proposed neural text matching models (section 4.3) are based on a Siamese [23] architecture, where both inputs undergo the same type of processing, regardless of the nature of the matching task. To take into account the differences between text matching tasks, we propose a framework that extends several state-of-the-art neural matching models, in order to enable them to handle the inputs in different ways w.r.t. the nature of the matching task. To do so, we use an attention-based model [31] to propose a matching architecture that gives more attention to the most important words in the input sequences. Subsequently, we extend several models from the state-of-the-art in the proposed framework.

7.2.1 The Asymmetry Aspect

In order to highlight the *asymmetry* aspect, we consider the *question-answer* matching task. Expressed in a natural language, the question describes a specific user's information need. It can be seen as a puzzle where the missing pieces must be found and put together, in a logical way, to solve the problem. The pattern of each gap in a puzzle describes the corresponding missing part. It is all the same for the question-answer matching problem, a pattern represents the missing information and it can be filled by one or more answers. Not all answers can be suitable, only those that conform to the pattern (the question) that describes the missing part can correctly fill it. The example in figure 7.1 from the WikiQA dataset illustrates this perception. In this figure, words of the answers *A1* and *A2* that are in **bold** characters represent semantic and syntactic relatedness with the question Q. The <u>underlined</u> words represent the keywords (pattern clues) in Q, and its corresponding matches in *A1* and *A2*. In this example, to correctly answer the question Q, we have to focus on the

terms *"How"* and *"immigrated to"* describing the missing information. Q is about how the immigration process was performed. The two different answers *A1* and *A2* give two different solutions, solution 1 and solution 2 respectively. Based on semantic and syntactic relatedness, both the answers match with the question. However, one can notice that solution 2 better solves the problem than solution 1 does. The answer *A2* contains the corresponding information. Hence, solution 2 represents the missing piece of the puzzle.

This example shows that syntactic and semantic-based similarities are not enough to completely solve the problem. The question has some words that require particular attention to retrieve the correct answer.

The previous example highlights the asymmetry aspect of the question-answer matching task as well as of similar problems, such as document-query matching. Hence, we consider the following definitions:

- The *symmetric matching* task consists in identifying if two texts are semantically similar, and concern inputs of the same type, such as sentence completion [262, 3], document classification [19, 263], and paraphrase identification [168, 20, 264].

- The *asymmetric matching* task consists of knowing if a text provides the information sought in another text. The input texts are supposed to be of different natures, and their similarity is determined not only by their semantic and lexical links but also by their complementarity. This type of tasks mainly includes question-answer matching [157, 22], document-query matching [185, 21], and textual entailment[1] [265].

7.2.2 The Asymmetry Sensitive Matching Architecture

The previous example in figure 7.1 shows the main idea about the *asymmetric* matching tasks, and the need to take into account the importance of each of the keywords of a text sequence, beyond the semantic and lexical links between the input sequences being matched. In this work, we propose an approach to take these aspects into account. Specifically, we use attention-based layers, to allow the matching model to focus on the most important words in the input sequences according to the nature of the task being addressed.

Model Description

We consider two generalized input sequences, $s_i^{(Q)} =< w_0^{(Q)} ... w_{|s_i^{(Q)}|}^{(Q)} >$ that can be a query, a question or any input sequence to be compared with another sequence; and $s_{ij}^{(D)} =< w_0^{(D)} ... w_{|s_{ij}^{(D)}|}^{(D)} >$ that can be a document, an answer sentence or any other sequence. Based on the general architecture that we previously described in figure 4.1, we define an extension framework where the model Φ is extended using attention layers in the function φ. This process is shown in figure 7.2, where we use Φ' instead of Φ and φ' instead of φ, to refer the extension of the original model Φ, the extended representation layer, respectively. φ' is supposed to process every input sequence in a different way according to the task being addressed.

[1] Inferring a directional relation between different text sequences.

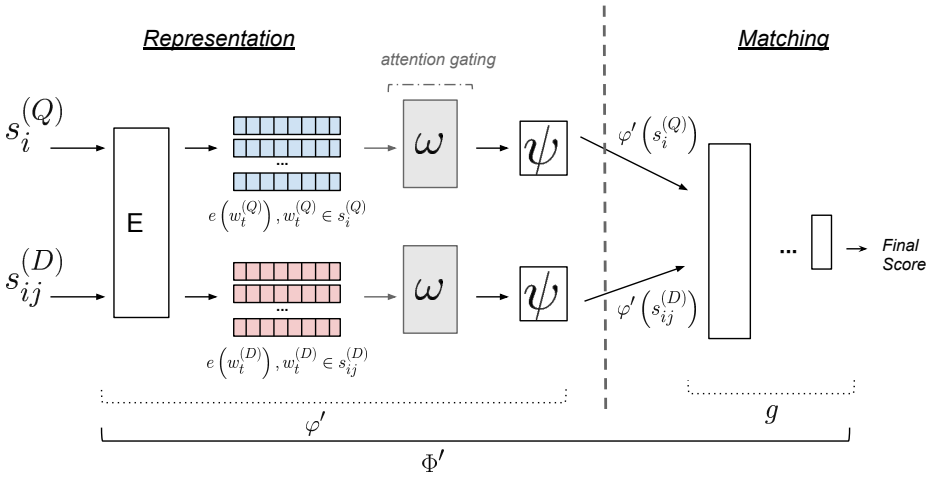


Figure 7.2: A generalized neural matching framework for extending state-of-the-art models with an attention gating layer.

In figure 7.2, the greyed layer ω represents the extension *attention* layer. Hence, $\varphi'\left(s_i^{(Q)}\right)$ and $\varphi'\left(s_{ij}^{(D)}\right)$ are representations corresponding to the input sequences $s_i^{(Q)}$ and $s_{ij}^{(D)}$, respectively. The representation function φ' is defined based on the attention layer ω with an activation parameter. More precisely, for an input sequence $s \in \left\{s_i^{(Q)}, s_{ij}^{(D)}\right\}$, an activation parameter $\rho^{(s)}$ is defined, and the corresponding representation $\varphi'(s)$ is then computed as in equation 7.1.

$$\varphi'(s) = \begin{cases} \psi \circ \omega(s) & \text{if } \rho^{(s)} = 1 \\ \varphi(s) & \text{otherwise} \end{cases} \tag{7.1}$$

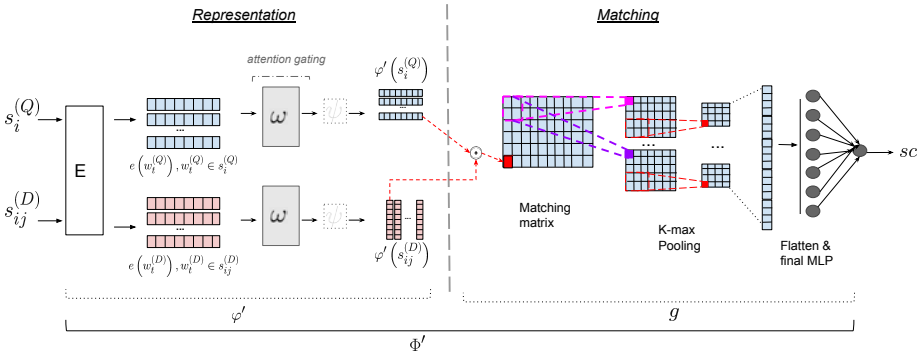
where φ is a function used to compute the internal representations of the inputs, as defined in equation 4.3. ψ is a representation function that combines the different word vectors $e(w_t)$ of s to compute a corresponding initial representation. The ω function computes a weighted representation for the input sequence, and is defined in equation 7.2.

$$\omega(s) = \bar{\alpha}s = \left[e(w_1)\,\alpha_1, ..., e\left(w_{|s|}\right)\alpha_{|s|}\right] \tag{7.2}$$

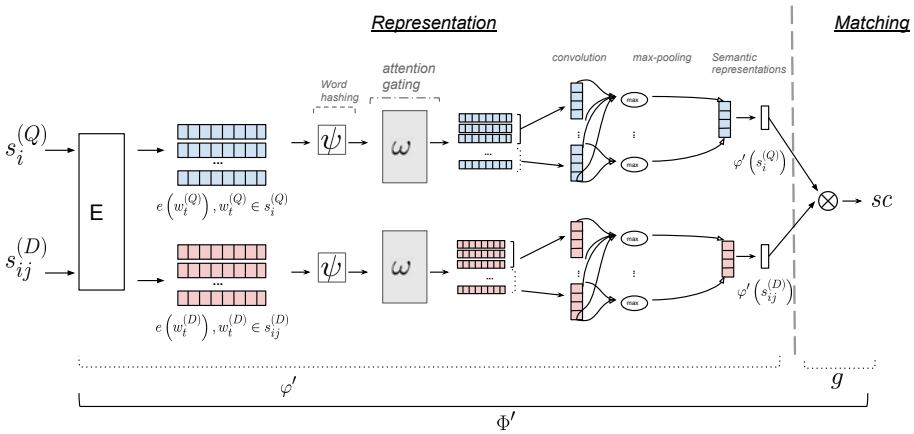
where w_t is the word at position t in s. $\bar{\alpha} = \left[\alpha_1, ..., \alpha_{|s|}\right]$ is the attention weight vector, where α_t is an attention weight computed as in equation 4.8.

The matching part of the framework described in figure 7.2 remains the same. The whole extended model Φ' is trained from end to end, where the corresponding parameters include both parameters of the original model Φ (figure 4.1) and parameters of the added *attention gating* layer. Thus, the parameters of Φ' are $\theta_{\Phi'} = \{\theta_\Phi, \theta_\omega\}$ with θ_Φ and θ_ω are the parameters of the original model Φ and those of the attention layer ω, respectively. Finally, the relevance score of an input sequence $s_{ij}^{(D)}$ w.r.t. a another input $s_i^{(Q)}$ is computed by the extended model Φ' defined as in equation 7.3.

$$\Phi' \stackrel{def}{=} g\left(\varphi'(s_i^{(Q)}), \varphi'(s_{ij}^{(D)})\right) \tag{7.3}$$



(a) Extension of the MatchPyramid model [19]



(b) Extension of the CDSSM model [185]

Figure 7.3: State-of-the-art models extension using the proposed attension-based framework.

The framework proposed in this section can be used to extend different state of the art models. In the following paragraph, we show some neural text matching models that we extended, based on the framework from figure 7.2.

Extension of Some State-of-the-art Neural Models

In section 7.2.2, we have described an extension framework that is applicable for most neural text matching model. We used a generalized input sequences, $s_i^{(Q)}$ and $s_{ij}^{(D)}$, to enable inclusion of the maximum number of text matching models. However, the adaptation of this framework could vary from model to another. This aspect is illustrated in figure 7.3 with two examples of extended neural text matching models. Namely, MatchPyramid [19] that is an interaction-focused model and CDSSM [185] that is a representation-focused model.

In figure 7.3a, we show our extended version of the MatchPyramid model [19]. In this case, the function ψ can be seen as a void function, where the inputs do not undergo any transformation. Hence, the matching part of this model uses the attention-based weighted word vectors instead of the original represen-

tations. The rest of the model remains unchanged, with a set of convolution and pooling layers that compute matching signals from different abstraction levels. In figure 7.3b, we show the extension of the CDSSM model [185]. In this figure, the ψ function corresponds to the *word-hashing* function [16], where tri-gram letters are used as an input instead of a sequence words. In this case, one can consider the original words of a sequence instead of the n-gram letters, by considering ψ as a void function, like in figure 7.3a. Thus, the ω layer compute attention weights for the n-gram letters or the original words. In both the extended models of figure 7.3, the function g represents the interaction part of each model, where the objective is to compute characteristics and intermediate results until the final output score sc.

We have also extended several other neural matching models, as described in figure 7.3, but we considered only two examples in this figure. All the models we have extended, including those shown in figure 7.3, are evaluated and compared to their original versions (section 7.5.1).

In the followings, we describe the experimental process and the evaluation results of our approach applied to different models. First, in the section 7.3 we describe the objective functions used for training. Then in section 7.4, we describe the experimental setup and discuss the different results.

7.3 Models training

In order to train the different neural models we used two different objective functions. In particular, the rank hinge loss and the categorical cross entropy.

7.3.1 Rank Hinge Loss

The rank hinge loss (RHL) is an objective function used to train a ranking tasks. This function is used in several text matching models [178, 266]. Given a sequence s and two other different sequences s^+ and s^-, such that s^+ is most similar to s and must be ranked better then s^-. Given a trainable model with θ a set of learnable parameters, the ranking loss function $\pounds_{RHL}$ is defined as in equation 7.4.

$$\pounds_{RHL}(s, s^+, s^-; \theta) = max(0, 1 - sc(s, s^+) + sc(s, s^-)) \tag{7.4}$$

where θ represents all the trainable parameters of the model being learned. $sc(s, s^*)$ is the matching score predicted by the model w.r.t. the input sentences s and s^*.

7.3.2 Categorical Cross Entropy

The categorical cross entropy (CCE) is an objective function used to train models designed for a classification tasks [267]. Formally, given two sequences s_1 and s_2, the objective of the cross entropy is to compute the probability that these sentences can be similar or of the same category. This means that it assess the probability $\hat{y}$ that the event $x = (s_1 \approx s_2)$ may occur. The loss function $\pounds_{CCE}$ is defined in equation 7.5.

$$\pounds_{CCE}(y, \hat{y}; \theta) = -\sum_x y \; log\,(\hat{y}) \tag{7.5}$$

Parameter	Value		Description
	WikiQA	QuoraQP	
num_iters	400	500	Number of training epochs.
test_weights_iters	400	500	Testing epoch.
text1_maxlen	20	20	Maximum length of the generalized query input.
text2_maxlen	40	20	Maximum length of the generalized document input.
batch_size	100	1024	Number of samples in one training batch.
losses	rank_hinge_loss	categorical_crossentropy	The objective function optimized during training.

Table 7.1: Descriptions and settings of the hyper-parameters of the MatchZoo [5] tool, in both WikiQA and QuoraQP datasets.

where $\hat{y}$ is the predicted probability computed by the model, and y is the *id* of the true label corresponding to x. θ represents all the trainable parameters of the model being trained.

7.4 Experiments

Experiments were performed[2] using the MatchZoo [5] framework for neural text matching models. We used two datasets, the WikiQA dataset [4] and the QuoraQP dataset. We provided more details about the used datasets and the different tools in chapter 5.

We adopted a cross-validation with 80% to train, 10% to test and 10% to validate the different models results. As embeddings, we used the publically available pre-trained 300-dimensional word vectors of GloVe[3], which is trained in a common crawl dataset. All the evaluated neural models were trained using the ranking hinge loss function (section 7.3.1) for 400 epochs, on the WikiQA dataset and the categorical cross entropy loss function (section 7.3.2) for 500 epochs, on the QuoraQP[4] dataset. For every model, we reported the results performance at the end of the training epochs. Table 7.1 show more details about the hyper parameters related to MatchZoo [5] and that are set to the same values regardless of the models being evaluated.

Concerning parameters of the baselines, we opted for the recommended hyper-parameters configuration, either on the corresponding papers or in Matchzoo[5]. Exceptionally, in the CDSSM model [185], we used embedded word vectors rather than the tri-letter hashing method [185] in order to assess the impact of the attention layer (equation 7.2) at the word-level, in the same way than in the other evaluated models.

[2]The corresponding code will be available on MatchZoo and public to allow the reproducibility of all the results we show in this chapter.

[3]http://nlp.stanford.edu/data/glove.840B.300d.zip

[4]The loss values of some of the models converged after more than 400 epochs in QuoraQP dataset.

[5]https://github.com/NTMC-Community/MatchZoo/tree/1.0/examples

7.5 Results and Analysis

In this section, we describe and analyze the different results of the proposed models in this chapter, separately, and compare the performance of each model to different baselines.

7.5.1 The Asymmetry Sensitive Approach Analysis

In this section, we evaluate and analyze the asymmetry-sensitive architecture that we propose for text matching. We use a set of labels to refer to every architecture configuration. The symmetric matching includes the basic model (Original) and the application of the attention layer ω (equation 7.2) to both model inputs simultaneously, we refer to this as (Q+A). The asymmetric matching includes applying the ω layer for one of the inputs at a time, the question only (Q) or the answer only (A).

As in the previous section, we answer a set of research questions related to the text matching problems, and how they could be solved with our architecture.

***RQ1:* What is the architecture's impact on matching text sequences of different natures?**

Table 7.2 shows the performance, in terms of MRR, P@1 and P@3, nDCG@1 and nDCG@3, of the different evaluated neural models in the WikiQA dataset, using the original architecture of each model as well as the extended architectures. The latter are noted by adding the labels (Q), (A) and (Q+A) as described above. In this table, the values in **bold** characters represent the maximum performances. The symbols ▲ and ▼ represent, the significance in[6],respectively, improvement and decrease of the performances.

We note that, for almost all models and metrics, at least one of the asymmetric architectures, either (Q) or (A), outperforms its symmetric counterparts, namely (Q+A) and the original version. Except the KNRM model, where the symmetric extended version (Q+A) outperforms all its asymmetric counterparts as well as the corresponding original model. Indeed, in the KNRM model [180] the strongest matching signals in the word-level translation matrix are fed for a simple MLP structure. This kind of networks does not enable the handling the different inputs individually. Hence, the attention weights computed in the word-level for both the input sequences, in the corresponding KNRM.(Q+A) extended model, gave better performance than the original KNRM does. Note that the improvements are significant for the MV-LSTM model. In table 7.2, the results show that taking into account the attention layer in (Q) or (A) allows a significant improvement in the performance of the different models, especially in terms of MRR and P@1 where: the architecture (Q) of the ARC-II, MV-LSTM and MatchPyramid models outperform the original architecture of each model. The architecture (A) of the DUET and CDSSM models largely exceeds their original architectures.

In order to analyze more closely the results shown in table 7.2, we want to check for improvements in the different questions of the *test* set of the WikiQA

[6]T-test with $p = 0.05$.

Model	MRR	P@1	P@3	nDCG@1	nDCG@3
ARC-I	0.6053	0.4430	0.2517	0.4430	0.5642
ARC-I.(Q)	0.5977	0.4346	0.2461	0.4346	0.5529
ARC-I.(A)	0.5913	0.4093	0.2560	0.4093	0.5671
ARC-I.(Q+A)	0.5931	0.4093	0.2531	0.4093	0.5656
ARC-II	0.5708	0.3840	0.2489	0.3840	0.5410
ARC-II.(Q)	0.5748	0.4177	0.2419	0.4177	0.5357
ARC-II.(A)	0.5528	0.3544	0.2531	0.3544	0.5327
ARC-II.(Q+A)	0.5814	0.3924	0.2503	0.3924	0.5485
CDSSM	0.5586	0.3671	0.2475	0.3671	0.5285
CDSSM.(Q)	0.5222	0.3333	0.2335	0.3333	0.4973
CDSSM.(A)	0.5886	0.4135	0.2461	0.4135	0.5490
CDSSM.(Q+A)	0.5622	0.3924	0.2405	0.3924	0.5266
DUET	0.6259	0.4599	0.2714	0.4599	0.6016
DUET.(Q)	0.6314	0.4641	0.2742	0.4641	0.6116
DUET.(A)	0.6383	0.4852	0.2714	0.4852	0.6112
DUET.(Q+A)	0.5982	0.4262	0.2531▼	0.4262	0.5589▼
KNRM	0.5208	0.3333	0.2264	0.3333	0.4788
KNRM.(Q)	0.5158	0.3038	0.2236	0.3038	0.4607
KNRM.(A)	0.5138	0.3249	0.2264	0.3249	0.4744
KNRM.(Q+A)	0.5282	0.3502	0.2321	0.3502	0.4942
MatchPyramid	0.6529	0.4726	0.2869	0.4726	0.6448
MatchPyramid.(Q)	**0.6715**	**0.5063**	**0.2981**	**0.5063**	**0.6649**
MatchPyramid.(A)	0.5575▼	0.3544▼	0.2531▼	0.3544▼	0.5381▼
MatchPyramid.(Q+A)	0.4698▼	0.2574▼	0.2110▼	0.2574▼	0.4211▼
MV-LSTM	0.6215	0.4388	0.2813	0.4388	0.6101
MV-LSTM.(Q)	0.6691▲	0.5021▲	0.2897	0.5021▲	0.6539▲
MV-LSTM.(A)	0.6174	0.4388	0.2714	0.4388	0.5984
MV-LSTM.(Q+A)	0.5904	0.4093	0.2517▼	0.4093	0.5562▼

Table 7.2: Comparison of the performance of the different evaluated models in the WikiQA dataset, using the different architectures.

dataset. Table 7.3 shows the performance of the different neural models that we extended and evaluated, using the corresponding different architectures, as compared to the classical BM25 and LM models. In this table, the extension "$.\omega$" refers to the application of the ω attention layer (equation 7.2) with the corresponding model, as described in the examples in figure 7.3. W/L/T represent the number of improved (Win), decreased (Loss) and unchanged (Tie) questions respectively, for each model as compared to its *Original* architecture. The symbols ▲ and ▼ represent, respectively the significant improvements and decrease in performance.

In table 7.3, we find that the MAP is better at least for one of the asymmetric architectures for all neural models, allowing better performance compared to the classical models. Regarding statistics on the different test questions, we note that the number of questions improved by the architectures (Q) and (A) is more important than with the architecture (Q+A), this is true for the different evaluated models. Note that these results strongly support our hypothesis regarding the impact of asymmetric architectures on question-answer asymmetric matching tasks. In addition, the performances obtained with the asymmetric model (Q)-MatchPyramid.ω are the best for this dataset[7], while the number of

[7](Q)-MatchPyramid.ω also outperforms all methods reported by MatchZoo in

Class	Model			MAP	W/L/T
Classical Models	BM25			0.5762	-/-/-
	LM			0.5932	-/-/-
Neural Models	Symmetric	Original	ARC-I	0.5792	-/-/-
			ARC-II	0.5606	-/-/-
			CDSSM	0.5473	-/-/-
			DUET	0.6113	-/-/-
			KNRM	0.5093	-/-/-
			MatchPyramid	0.6436	-/-/-
			MV-LSTM	0.6046	-/-/-
		(Q+A)	ARC-I.ω	0.5829	57/61/119
			ARC-II.ω	0.5648	66/66/105
			CDSSM.ω	0.5523	85/76/76
			DUET.ω	0.5801▼	64/76/97
			KNRM.ω	0.5198	83/77/77
			MatchPyramid.ω	0.4697▼	56/129/52
			MV-LSTM.ω	0.5562	64/83/90
	Asymmetric	(Q)	ARC-I.ω	0.5792	37/45/155
			ARC-II.ω	0.5595	42/72/123
			CDSSM.ω	0.5134	76/91/70
			DUET.ω	0.6158	69/58/110
			KNRM.ω	0.4982	76/71/90
			MatchPyramid.ω	**0.6591**	47/42/148
			MV-LSTM.ω	0.6507▲	70/41/126
		(A)	ARC-I.ω	0.5815	58/61/118
			ARC-II.ω	0.5439	60/81/96
			CDSSM.ω	0.5779	78/75/84
			DUET.ω	0.6251	68/53/116
			KNRM.ω	0.5015	77/76/84
			MatchPyramid.ω	0.5502▼	43/97/97
			MV-LSTM.ω	0.6165	68/67/102

Table 7.3: Comparison of the performances, in terms of MAP, of the *Symmetric* and *Asymmetric* matching of the different neural models compared to classical models, in the WikiQA collection.

improved queries is greater for the CDSSM model with architecture (Q+A). In table 7.3, we note that when the numbers of improved and deteriorated questions are very close, the difference between the performance of the original model and its corresponding extended architecture, such as (Q), (A) or (Q+A) is not significant.

In order to get more explanations about these results, we also analyzed the questions whose results were improved or deteriorated to see if there is a correlation with the type of question (who, when, where...). Hence, another research question needs to be answered at this point. This question concerns the correlation between the type of questions addressed by the models and the improvements obtained with their extended architectures.

***RQ2:* Is there any correlation between the question types and the architecture that gets better improvement?**

In order to answer this question, we first analyze the different questions in the WikiQA test dataset. There are five different question types: *who*, *when*, *what*, *where*, and *how*. Figure 7.4 shows the distribution of the different types over

https://github.com/NTMC-Community/MatchZoo.

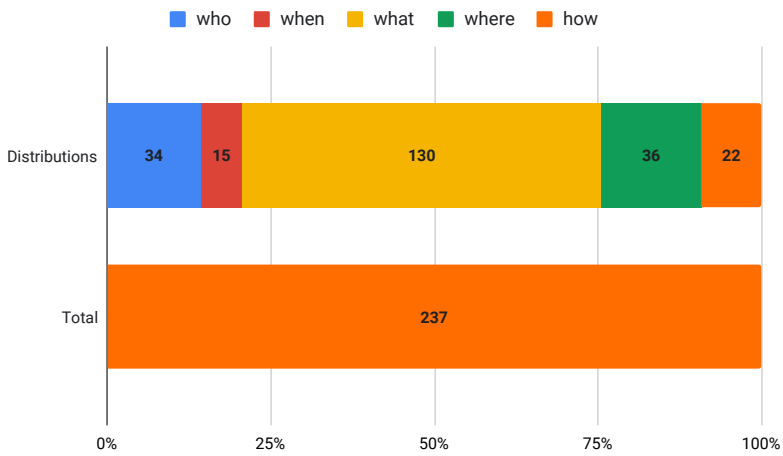


Figure 7.4: Distribution of the different question types in the *test* dataset of WikiQA.

the total of 237 test questions. In this figure, we notice that almost 55% of the test questions are *what* questions. The remaining 45% is distributed among the other four different types, and only 15 questions are of type *when*.

Given the non-uniform distribution of the different types of questions, in the test corpus of WikiQA dataset, as described in figure 7.4, it doesn't seem obvious to figure out the type of questions for which every model is more effective. For instance, if a model improves 10 questions of type *who* and 10 questions of type *what*, we cannot figure out in which type of the two this model is more efficient. Hence, we analyzed the W/L/T variations of the performance, in terms of MAP[8], for the different extended models compared to their original counterparts, for each type of question. The objective of this analysis is to find out, for every extended model, the type of questions on which it is most effective. Table 7.4 shows the proportions W/L/T of every extended model. In this table, values in **bold** characters correspond to the maximum Win (W) value per model.

In table 7.4, we notice that every model extension is more effective on one or more question type than its corresponding counterparts. For instance, ARC-II (Q) is more effective on *where* questions. Its counterpart ARC-II (A) is more effective on *how* questions, and finally ARC-II (Q+A) is more effective on *who* questions. This behaviour is also noticeable with the other models, in particular DUET, KNRM, and MV-LSTM, compared to their corresponding extensions. Based on this, we approximated the results of an ideal model that can automatically adapt its configuration, either (Q), (A) or (Q+A), depending on the type of question being processed. This corresponds to the *oracle* version of every evaluated model, whose performances are given in table 7.5. Formally, given a model Φ whose extensions are $\Phi_{(Q)}$, $\Phi_{(A)}$ and $\Phi_{(Q+A)}$. The performances in terms of an evaluation measure M of the corresponding *oracle* model Φ_{oracle}

[8] We only considered the MAP measure since it gives an overall evaluation of all the test set of questions. But, one can consider another measure, such as nDCG@k, precision or else, for the same analysis.

	Proportions %														
Question Type	where			how			who			when			what		
Model	W	L	T	W	L	T	W	L	T	W	L	T	W	L	T
ARC-I (Q)	9.1	22.7	68.2	16.7	33.3	50.0	14.7	8.8	76.5	**26.7**	6.7	66.7	15.4	18.5	66.2
ARC-I (A)	31.8	13.6	54.5	30.6	30.6	38.9	14.7	17.6	67.6	**33.3**	13.3	53.3	23.1	30.0	46.9
ARC-I (Q+A)	27.3	18.2	54.5	27.8	33.3	38.9	17.6	14.7	67.6	**40.0**	13.3	46.7	22.3	29.2	48.5
ARC-II (Q)	**27.3**	18.2	54.5	19.4	33.3	47.2	17.6	8.8	73.5	13.3	53.3	33.3	16.2	34.6	49.2
ARC-II (A)	27.3	18.2	54.5	**33.3**	47.2	19.4	26.5	17.6	55.9	26.7	20.0	53.3	22.3	39.2	38.5
ARC-II (Q+A)	27.3	27.3	45.5	27.8	27.8	44.4	**32.4**	11.8	55.9	26.7	33.3	40.0	26.9	31.5	41.5
CDSSM (Q)	31.8	31.8	36.4	38.9	41.7	19.4	20.6	35.3	44.1	**46.7**	26.7	26.7	31.5	40.8	27.7
CDSSM (A)	**54.5**	27.3	18.2	38.9	36.1	25.0	17.6	29.4	52.9	33.3	26.7	40.0	30.8	33.1	36.2
CDSSM (Q+A)	**50.0**	27.3	22.7	38.9	27.8	33.3	23.5	17.6	58.8	33.3	40.0	26.7	36.2	36.9	26.9
DUET (Q)	**36.4**	18.2	45.5	33.3	30.6	36.1	26.5	14.7	58.8	26.7	33.3	40.0	27.7	25.4	46.9
DUET (A)	**40.9**	18.2	40.9	30.6	33.3	36.1	26.5	17.6	55.9	40.0	20.0	40.0	25.4	21.5	53.1
DUET (Q+A)	31.8	22.7	45.5	**36.1**	38.9	25.0	26.5	23.5	50.0	26.7	40.0	33.3	23.8	33.1	43.1
KNRM (Q)	**45.5**	13.6	40.9	44.4	27.8	27.8	29.4	20.6	50.0	13.3	40.0	46.7	29.2	34.6	36.2
KNRM (A)	36.4	31.8	31.8	22.2	33.3	44.4	35.3	23.5	41.2	**40.0**	20.0	40.0	33.1	35.4	31.5
KNRM (Q+A)	40.9	18.2	40.9	**41.7**	25.0	33.3	35.3	29.4	35.3	20.0	46.7	33.3	33.8	36.2	30.0
MatchPyramid (Q)	**36.4**	4.5	59.1	22.2	16.7	61.1	17.6	17.6	64.7	6.7	33.3	60.0	18.5	18.5	63.1
MatchPyramid (A)	**31.8**	36.4	31.8	19.4	55.6	25.0	8.8	38.2	52.9	0.0	53.3	46.7	20.0	36.9	43.1
MatchPyramid (Q+A)	**31.8**	54.5	13.6	16.7	63.9	19.4	17.6	52.9	29.4	13.3	66.7	20.0	26.9	50.8	22.3
MV-LSTM (Q)	27.3	18.2	54.5	25.0	16.7	58.3	20.6	17.6	61.8	26.7	20.0	53.3	**33.8**	16.9	49.2
MV-LSTM (A)	31.8	22.7	45.5	**36.1**	30.6	33.3	20.6	26.5	52.9	20.0	46.7	33.3	29.2	26.9	43.8
MV-LSTM (Q+A)	22.7	27.3	50.0	**38.9**	30.6	30.6	14.7	32.4	52.9	20.0	46.7	33.3	28.5	36.9	34.6

Table 7.4: Comparison of W/L/T variations of the performance, in terms of MAP, of the extended models compared to their original counterparts, w.r.t. every question type in the WikiQA dataset.

are estimated as in equation 7.6.

$$M(\Phi_{oracle}) \geq max_M \left(M(\Phi), M(\Phi_{(Q)}), M(\Phi_{(A)}), M(\Phi_{(Q+A)})\right) \tag{7.6}$$

where $M(x)$ gives the performance value of the model x according to the evaluation measure M. In table 7.5, the underlined values refer to the best achieved by the models in their original versions. The values in **bold** characters refer to the best performance w.r.t. every measure, and symbols ▲ the significance of the improvements according to the *t-test*. In this table, we notice that all the *oracle* versions perform better than the original models with significant improvements. These results approximate the best results that could be produced by a model that can automatically adapt its configuration, specifically to be symmetric (A) or (Q), or asymmetric (Q+A) or stay in the original one, according to the nature of the matching task being addressed. Constructing this model represents the aim of a future work.

***RQ3:* What is the impact of the attention layer on the behaviour of the different neural models?**

To answer this question, we first analyze the behavior of the different neural models, with and without the attention layer, w.r.t. the ranking of the different candidate answers of a given question from the dataset. Next, we analyze attention weights computed by this layer for every word in an example question and/or answer. Let us consider the following question, from the WikiQA dataset, and for which there are 10 candidate answers with only 4 of them are relevant.

q: "What happened to George O'malley on grey's anatomy?"

In figure 7.5, we represent the ranking of the 10 candidate answers using a list of 10 squares. Each square represents one of the candidate answers. Relevant answers are represented by opaque squares. The objective is to push the relevant

Version	Models	Performances					
		MAP	P@1	P@3	nDCG@1	nDCG@3	MRR
Original	ARC-I	0.5879	0.4430	0.2517	0.4430	0.5642	0.6053
	ARC-II	0.5606	0.3840	0.2489	0.3840	0.5410	0.5708
	CDSSM	0.5473	0.3671	0.2475	0.3671	0.5285	0.5586
	DUET	0.6113	0.4599	0.2714	0.4599	0.6016	0.6259
	KNRM	0.5093	0.3333	0.2264	0.3333	0.4788	0.5208
	MatchPyramid	0.6443	0.4726	0.2869	0.4726	0.6448	0.6529
	MV-LSTM	0.6046	0.4388	0.2813	0.4388	0.6101	0.6215
Oracle	ARC-I.ω	0.6575▲	0.5148▲	0.2883▲	0.5148▲	0.6556▲	0.6737▲
	ARC-II.ω	0.6701▲	0.7089▲	0.3024▲	0.5190▲	0.6764▲	0.6854▲
	CDSSM.ω	0.7180▲	0.5992▲	0.3136▲	0.5992▲	0.7220▲	0.7349▲
	DUET.ω	0.7354▲	0.6203▲	0.3136▲	0.6203▲	0.7350▲	0.7485▲
	KNRM.ω	0.6629▲	0.5190▲	0.2939▲	0.5190▲	0.6525▲	0.6784▲
	MatchPyramid.ω	**0.7726**▲	0.6456▲	**0.3375**▲	0.6456▲	**0.7810**▲	**0.7862**▲
	MV-LSTM.ω	0.7569▲	**0.6540**▲	0.3263▲	**0.6540**▲	0.7677▲	0.7778▲

Table 7.5: Results of the *oracle* version of every evaluated mode, compared to the corresponding original ones.

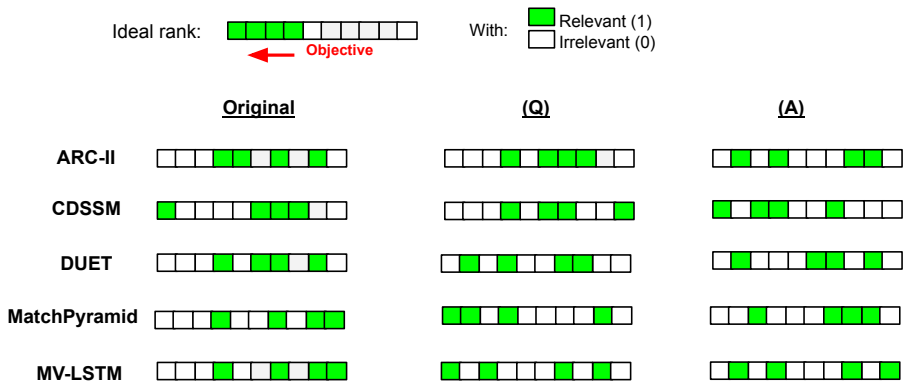


Figure 7.5: Ranking of the different candidate answers corresponding to the question example, by some of the evaluated models, in their different versions.

answers to the left of the list in order to return them first, which corresponds to the *Ideal rank* result. In figure 7.5, we considered some of the evaluated models, without and with the attention layer ω (equation 7.2), used to construct the different extensions, (Q) and (A).

We consider the previous question q, and the answer a which one of the corresponding relevant answers, and which is returned by all the neural models considered in figure 7.5. For both q and a, we give the corresponding list of keywords indexed in MatchZoo:

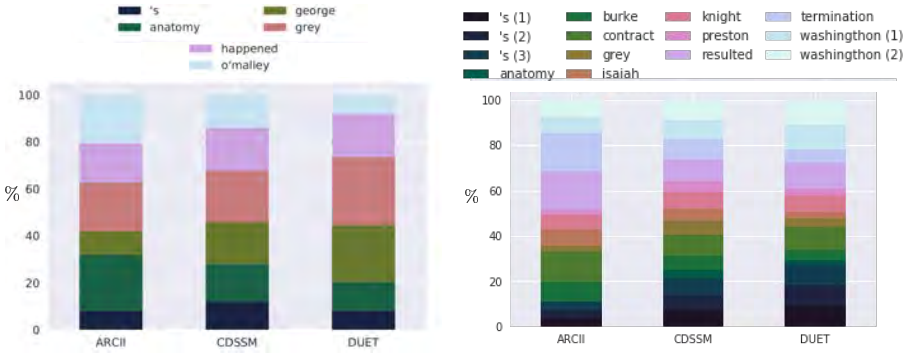
q: "What happened to George O'malley on Grey's anatomy?"

keywords: happened, george, o'malley, grey, 's, anatomy.

a: "In 2007, Knight's co-star Isaiah Washington (Preston Burke) insulted him with a homophobic slur, which resulted in the termination of Washington's Grey's Anatomy contract."

keywords: knight, 's (1), isaiah washington (1), preston, burke, resulted, termination, washingthon (2), 's (2), grey, 's (3), anatomy, contact.

For each of the neural models considered in figure 7.6, we retain the asymmetric



(a) Weights of the keywords of the question q computed by the attention layer ω of each model with its corresponding asymmetric architecture (Q).

(b) Weights of the keywords of the question a_i computed by the attention layer ω of each model with its corresponding asymmetric architecture (A).

Figure 7.6: Comparison of importance weights computed by the attention layer of the asymmetric architectures of three different models.

architecture with which the model performed best in previous analyses, and then investigate the weights of importance computed in each model, for the keywords of the question q and the answer a. The figure 7.6 shows the different computed values. The objective is to analyze how the attention layer weights the different words according to their real (literal) importance in the corresponding sentence.

In the figure 7.6a, the keywords *happened* and *george* are highly weighted in the question q by all the three models, which is consistent with the objective of q. Indeed, the question is about what happened to *george*. In figure 7.6b, the keywords *contract*, *resulted* and *termination* have acquired highest weights. These words represent the core information provided by the answer a. These results show the contribution of using the attention layer in the weighting process of the words of the question/answer according to their real importance.

RQ4: What is the impact of the asymmetric architecture on the matching of texts of the same nature?

So far, we have analyzed the contribution of the architecture we propose (section 7.2.2) in the case of asymmetric matching tasks, such as question and answer matching. We also want to analyze the interest of our approach in the case of symmetric matching tasks. We consider the paraphrase identification task. Specifically, using the QuoraQP dataset (section 5.2), we are going to analyze the performance of our model in detecting similar question pairs.

In figure 7.7, we show the performances in terms of *Accuracy* of different extended neural models while matching question pairs in the QuoraQP dataset. In this figure, we find that, as expected, most of the models do not benefit from the asymmetric architectures, specifically (A) and (Q). This is due to the symmetric nature of this matching task, except the CDSSM model for which performances of the asymmetric architectures, (A) and (Q), substantially exceed those of the original version. Indeed, the CDSSM model uses a convolution layer to take into account the context [185] of a word, and the attention weights computed

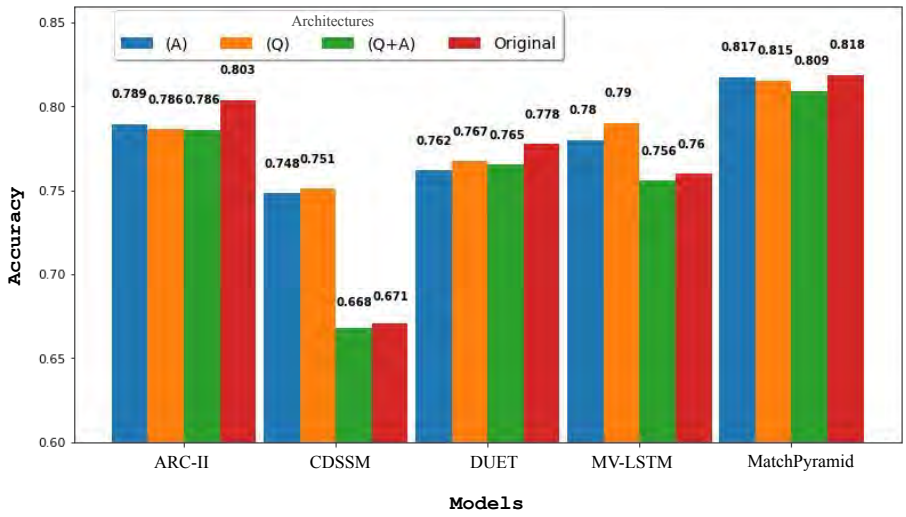


Figure 7.7: Performance, in terms of *accuracy* in the QuoraQP dataset, of different extended neural models using different architectures.

exclusively for one of the inputs at a time enable to inform the model about the most important words of each context, thus improving its performance. Similarly in case of the MV-LSTM model, where words are handled depending on their positions [3], attention weights are used to inform the model about the importance of the word at each position.

For the rest of the models, the asymmetric architectures have not a significant impact on their results, and the corresponding original architectures, ARC-II, DUET, and MatchPyramid, perform better than their extended counterparts.

Given the results shown in figure 7.7, we want to analyze the behaviour of our asymmetric matching architecture in the symmetric matching task. To do so, we consider the following three different example questions from the QuoraQP dataset: q_1: *"Are we living in a simulation?"*
q_2: *"Do we have any proof that we live in a simulation?"*
q_3: *"Is there a possibility that we actually are existing in a programmed platform?"*
Where, q_1 is similar to q_2 (Label = 1), but not similar to q_3 (Label = 0). In the table 7.6 we present the similarity scores computed by the different models using each of the different architectures. In table 7.6, opaque green cells refer to the improvement of the similarity score of the positive pair (q_1, q_2) with the corresponding model architecture, compared to the original architecture of that model. We note that the score of the question q_2 is optimized in the asymmetric architectures (Q) and (A) as well as in symmetric architecture (Q+A). In particular, in case of the ARC-II and CDSSM models, the original architecture assigns inappropriate scores to the different pairs (q_1, q_2) and (q_1, q_3), the attention layer has allowed to compute similarity scores that correctly distinguish the positive pair (q_1, q_2) from the negative one (q_1, q_3). To verify these results, we computed the attention weights assigned by the ω layer, applied for q_1 in the

Question Pairs		Label	Architecture	Predicted Score $sc(q_i, q_j)$				
				ARCI-II	CDSSM	DUET	MatchPyramid	MV-LSTM
q_1	q_2	1	Original	0.0464	0.3694	0.7067	0.1002	0.5658
	q_3	0		0.9641	0.3694	0.1416	0.0131	0.1447
	q_2	1	(Q+A)	0.3621	0.2092	0.6211	0.9084	0.6783
	q_3	0		0.3464	0.2917	0.0840	0.0456	0.0150
	q_2	1	(Q)	0.2638	0.7184	0.6946	0.7736	0.7060
	q_3	0		0.0024	0.2727	0.1576	0.1612	0.0162
	q_2	1	(A)	0.1530	0.6922	0.8057	0.6116	0.6600
	q_3	0		0.0462	0.2208	0.4703	0.1055	0.2225

Table 7.6: The similarity scores assigned to a pair of similar questions (q_1, q_2) and a pair of non-similar questions q_1, q_3) by the different neural models within the symmetric and asymmetric architectures.

architectures (Q) and (Q+A), and those computed for the other two questions, q_2 and q_3, in the architectures (A) and (Q+A). The figure 7.8 shows the attention weights assigned to each of the terms of these questions by the different models.

In figure 7.8, we notice that for each model, either the symmetric (Q+A) or asymmetric (Q) and (A) architecture is used, the attention layer has the same impact on the computation of the importance weights for the different words of the three questions q_1, q_2 and q_3. In addition, for the models that compute incorrect similarity scores (table 7.6), such as ARC-II and CDSSM, the attention layer allows the model to focus on the keywords of the different inputs. Therefore, these models produced more suitable similarity scores, using the asymmetric architectures (A) and (Q), as shown in table 7.6.

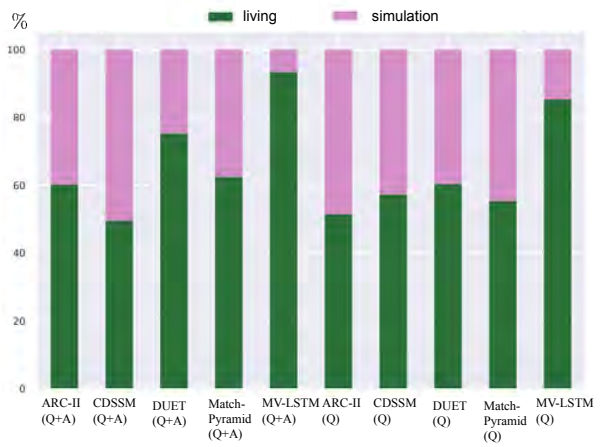
***RQ5:* How effective is our approach compared to state-of-the-art models that are not Implemented in MatchZoo?**

We would like to compare our results to others models that already performed good results in the datasets that we used in these experiments. Hence, we considered external models to the MatchZoo tool, and compared the best performances that we obtained in our experiments, with the asymmetric/symmetric architectures, to those published in the corresponding papers to the considered baselines. These baselines are:

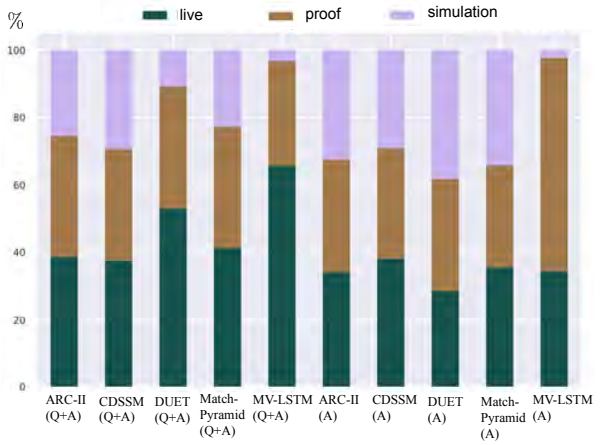
- $SUM_{BASE;PTK}$ [268] is a Kernel-based model, combining intra and cross word similarities between text pairs. The intuition behind this model is that similar questions are likely to require similar answer patterns. Hence, the model computes similarities between different training pairs, and use them as additional features.

- $PairwiseRank + SentLevel$ [231] is an approach for answer selection for question answering. This method directly exploits existing pointwise neural network models, by extending the Noise Contrastive Estimation approach with a triplet ranking loss function, in order to exploit interactions in triplet inputs, containing a question paired with positive and negative examples.

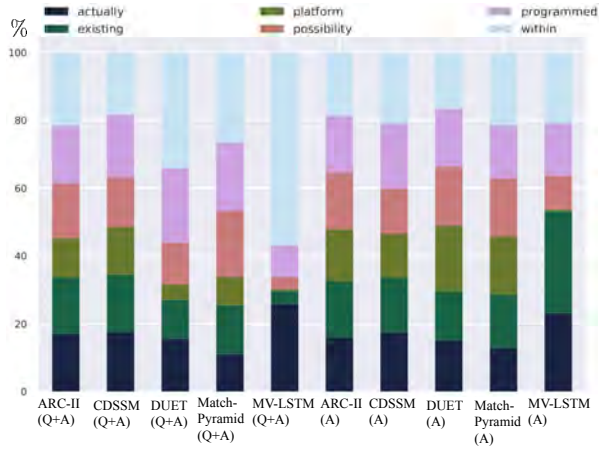
Table 7.7 shows the performance of our approach in terms of MAP and MRR. *Internal* models are the models implemented in MatchZoo, for each model,



(a) q_1



(b) q_2



(c) q_3

Figure 7.8: Proportion (%) of the importance weights of the keywords of three different questions, from the QuoraQP dataset, computed by the attention layer of the extended models.

Model		MAP	MRR
Internal (extended models)	ARC-II$_{(Q)}$	0.5595	0.5748
	CDSSM$_{(A)}$	0.5779	0.5886
	DUET$_{(A)}$	0.6251	0.6383
	MatchPyramid$_{(Q)}$	0.6591	0.6715
	MV-LSTM$_{(Q)}$	0.6507	0.6691
External	$SUM_{BASE;PTK}$ [268]	**0.7559**	**0.7700**
	$PairwiseRank + SentLevel$ [231]	0.7010	0.7180

Table 7.7: Comparison of the results of our approach, using *Internal* models at MatchZoo, against *External* models.

we consider the asymmetric architecture that gave the best results, specifically (Q) or (A). The values in **bold** characters show the best performance for each measure. *External* models represent state-of-the-art models that are not implemented in MatchZoo. In this table, we note that the external model $SUM_{BASE;PTK}$ gives better results than the other models. In addition, the $PairwiseRank + SentLevel$ model gives better results than the different *Internal* models with the corresponding architectures. Future work will focus on evaluating our asymmetric approach with these models in order to obtain new state-of-the-art results.

7.5.2 Position VS Attention

In this section, we aim at analyzing the main impact of word-level attention weights combined with the position feature of the different words. To do so, we consider MV-LSTM [3], which is a position based model for sequence matching, and that we extended and evaluated in the previous analysis. We aim to answer two main research questions:

***RQ6:* How does the position-based model behave with/without the attention weights?**

Table 7.8 shows the 3 first answers, corresponding to one sampled question from WikiQA dataset. The considered question is *"Where do crocodiles live?"* and there are 21 possible answers in the WikiQA dataset, where only one question is relevant ($label = 1$).

In this table, we show how these answers are ranked by the proposed architectures of the MV-LSTM model. The number between brackets (.) corresponds to the retrieval rank according to every model. Note that the answers retrieved by the MV-LSTM model [3] as well as those retrieved by the different MV-LSTM model architectures, deal with the same subject as the question. However, the correct answer is ranked better by the MV-LSTM (Q) model. Compared to the rank with MV-LSTM (A) and MV-LSTM (Q+A) models. This example shows the advantage of focusing on the input question words.

First answers	Predicted scores and rank values (.)				True *label*
	MV-LSTM	Extended			
		(Q)	(A)	(Q+A)	
Crocodiles (subfamily Crocodylinae) or true crocodiles are large aquatic tetrapods that live throughout the tropics in Africa, Asia, the Americas and Australia.	(3) 0.3646	(1) 0.9003	(2) 0.2871	(3) 0.3640	1
Crocodiles have more webbing on the toes of the hind feet and can better tolerate saltwater due to specialized salt glands for filtering out salt, which are present but non-functioning in alligators.	(1) 0.8989	(2) 0.8826	(1) 3.5154	(1) 2.0885	0
Also when the crocodile 's mouth is closed, the large fourth tooth in the lower jaw fits into a constriction in the upper jaw.	(2) 0.6290	(4) -1.1083	(4) -0.2245	(6) -0.2575	0
They are carnivorous animals, feeding mostly on vertebrates such as fish, reptiles, birds and mammals, and sometimes on invertebrates such as molluscs and crustaceans, depending on species and age.	(6) -2.0848	(3) -0.6799	(3) 0.2831	(2) 0.5409	0

Table 7.8: Ranking of the 3 first answers retrieved by the MV-LSTM model, among 21 possible answers corresponding to question *"Where do crocodiles live?"* in WikiQA dataset, compared to the extended versions using attention features.

***RQ7:* What is the gain of the attention weights combined with the position features in a classification task?**

Figure 7.9 shows the comparison of the accuracy evolution during the training and validation phase of the different architectures of the proposed model MV-LSTM and the MV-LSTM baseline, in the QuoraQP dataset. The red line on the graphics corresponding to MV-LSTM (A), MV-LSTM (Q) and MV-LSTM (Q+A), show the accuracy value of the MV-LSTM model [3] after training.

Note that all MV-LSTM architectures evolve in the same way and and get higher accuracy values compared to the MV-LSTM performance. MV-LSTM (Q) outperforms the MV-LSTM baseline with 3.8% in terms of accuracy. The attention-based weights has the same effect in the different MV-LSTM architectures, in the QuoraQP dataset, because it is a dataset for a symmetric matching task, where inputs are of the same nature, where the task consists of identifying whether an input question is a duplicate of another question or not. So, the comparison process considers inputs with approximately a same length and the same structure.

7.6 Conclusion

In this chapter, we presented our contributions to the neural short text matching state-of-the-art. The experimental results support our hypothesis concerning the impact considering the *asymmetry* aspect of several text matching tasks, in the architecture of the matching model; and showed the advantage of combining the position features and and the word-level attention weights.

First, we highlighted the asymmetry aspect of several text matching tasks, and then defined a generalized architecture enabling the model to process the in-

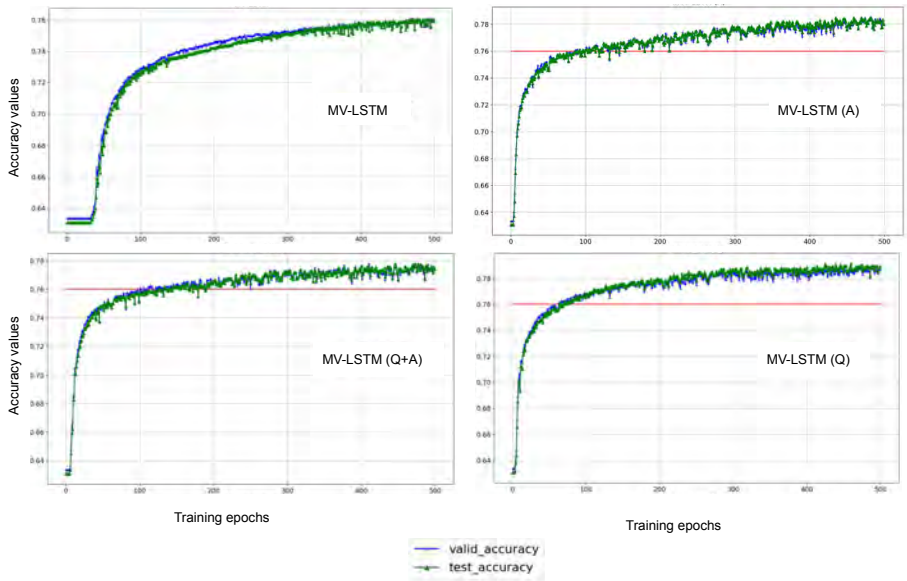


Figure 7.9: Comparison of the *valid* and *test accuracy* values during training on the QuoraQP dataset, of the MV-LSTM [3] model with/without the attention layers.

put text sequences w.r.t. their nature. In a second step, we extended several well known text matching models and built their corresponding asymmetric architectures. We performed experiments in two different question-answer datasets, considering the question-answer matching which is an asymmetric task, and the paraphrase identification which is a symmetric matching task. The results we obtained are promising in addressing the asymmetry aspect thanks to the attention layer. In the case of symmetric tasks, we saw that the asymmetric architecture did not have a significant impact on the performance of some models (*RQ4*). The attention layer corrected the similarity scores computed by models based on an MLP architecture, such as the CDSSM model [185], and the MV-LSTM model [3] based on positional features of words. In this model, the attention coefficients combined with the position information helped improving the results with more than 3.8% in terms of accuracy.

Finally, the analysis (*RQ2*) that we performed, concerning the questions type, have revealed that there is an important correlation between the type of a question and the extended models performances. Besides, the estimated *oracle* results are more performent than those of all the evaluated matching models. Hence, future work will focus our reflection on a model that, in one hand handle the question type automatically, and in the other hand take into account the symmetric/asymmetric aspect of the matching task.

CHAPTER 8

ATTENTION-BASED MULTI-LEVEL RELEVANCE ASSESSMENT

8.1 Introduction

Dealing with the document length is one of the most important challenges in IR [269]. The main problem is that, long documents may cover different topics, and only some parts are relevant to the query. This may complicate the matching task and decrease performances of current neural IR models. To deal with this issue, common neural IR models define a maximum document length in advance and pad shorter documents [21]. However, cutting off the exceeded text of a document could lead to information loss. Pang et al. [21] showed that the exceed part of a long document usually contains much important matching information. To highlight this aspect, we consider the query number 253 of the AP TREC dataset. In this dataset, there are six relevant documents, of which the longest contains more than 600 words and the shortest contains about than 250 words. Hence, it becomes difficult to set a maximum length for all documents in this dataset for a neural matching model.

Figure 8.1 shows an explicit description[1] of the relevance signals distribution in a query-document matching matrix, corresponding to the query 253 (y axis), containing three words, and the longest relevant document (x axis) containing more than 600 words. In this figure, note that the strongest matching signals, corresponding to similarity values close to $max = 1.0$ and that are highlighted with the white rectangles, are located after the 250th (around the 270th) word, and several other strong matching signals appear in the second half part of the document (300th to the 600th word). This example highlights the fact that a query-document matching information can be lost if the document content is truncated. Another study is provided by Fan et al. [27], where the authors discussed the diverse relevance granularity in a long document, and that could be relevant completely or include several relevant passages or paragraphs.

[1] This example is realized by using the word-word cosine similarities, between all the words of a TREC query and all the words of one of the corresponding relevant documents in the AP dataset.

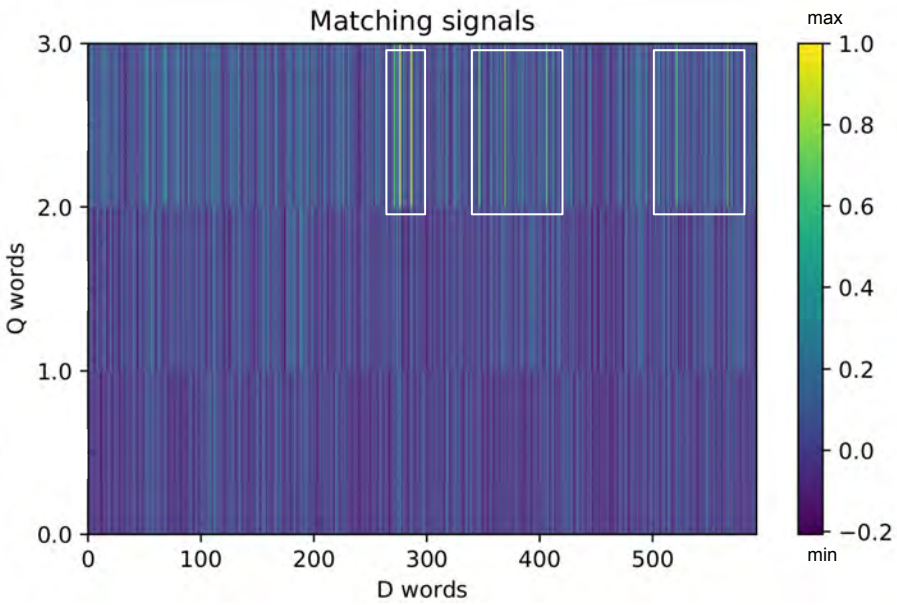


Figure 8.1: Example of a matching signals distribution over a relevant long document w.r.t. a query of three words.

In this chapter, we first revisit the question of whether passage-based methods could help neural models to overcome the document length problem or not. Several passage-based models [28, 29, 30] have been already proposed in IR to rank long documents. The main limitation of the previously proposed methods is that passage-level signals are combined in a simple way that higher level relevance patterns are not considered. For instance, the best-passage approach [29] that rank the whole document based on its most relevant passage, or the linear combination-based model [28] that merely aggregate the scores of the different passages in a linear function. These methods cannot capture complex relevance information, such as the interaction between passages, which could help to figure out if the document is truly relevant or not w.r.t. the query. Differently from previous work [29, 28, 27], we assume that some passages of a long document are likely to be more relevant than others and need to be considered differently, and hence put more attention on them. We focus on the study of two main aspects: (1) the impact of considering a document passages, in order to capture and combine the relevant parts of a long document, and solve the document length problem, (2) the impact of putting more attention on some passages of the same document and its component words, and words of the query.

In the followings, we first (section 8.2) define a multi-level assessment framework based on attention layers, in both word and passage levels, to extend several neural matching models from the state-of-the-art. The input document is represented with a set of passages that are likely to be relevant. Furthermore, (section 8.3) we propose our attention-based multi-level matching model (aM3), where we consider the document relevance at three different levels, namely at the *word-level*, the *passage-level* and the *document-level*.

8.2 A Multi-level Attention-based Architecture

In this section, we present a multi-level attention-based approach to assess a document relevance. The main goal is to evaluate the impact of putting more attention in some words of a query or a document, and in some passages of the document. We present in the following a general framework that can be used to extend several neural models, as described in figure 8.2. First, we introduce the document representation, then we describe our multi-level attention approach.

8.2.1 Passage-based Document Representation

To deal with the document length problem in neural IR models, we propose to represent a document as a set of passages of a long document to construct its representation. Specifically, Every document in the dataset could be segmented into passages of k words, and represented by $D = \{P_1, ..., P_l\}$ which is a set of l different passages from this document, where every passage $P_i \stackrel{def}{=} < w_{ij}^{(D)} ... w_{ij+k-1}^{(D)} >$ is a sequence window of k words, and $w_{ij}^{(D)}$ is a word at position j of the passage P_i.
This representation can be used with different neural matching models from the state-of-the-art. In the following, we describe how the attention weights can be put to highlight a different passages and their component words in a document, as well as in the query words.

8.2.2 The Multi-level Attention

We assume that in a long document, there are some passages that are more relevant that others for a given query. Therefore, we need to consider them differently. In addition, words of every text (query and passages) are not of the same importance, and we need to put more attention in the most important ones. To do so, we propose to extend several state-of-the-art neural text matching models by adding attention layers. More specifically, given a matching model Φ of the general framework defined in chapter 4 (figure 4.1), where the inputs $s_i^{(Q)}$ and $s_{ij}^{(D)}$ refer to the query Q and a passage of a document D, respectively. Φ can be extended (construct model Φ') using attention layers. These layers can be put in the word-level and/or to the passage-level, corresponding to the layers ω and ω_p, respectively in figure 8.2.

In this figure, we consider a query of three words and passages of three words each one. These words are weighted by the layer ω as defined in equation 7.2. At the passage-level, the layer ω_p computes attentions weights for the different passages after aggregating their word vectors, as defined in equation 8.1.

$$\omega_p \left(P_1, ..., P_l\right) = \alpha^{(P)} \varphi'(D) \tag{8.1}$$

where $\alpha^{(P)} = [\alpha_1^{(p)}, ...\alpha_l^{(p)}]$ is the attention weight vector corresponding to the different passages in the input document D. Parameter $\alpha_i^{(p)}$ is the attention weight corresponding to the passage P_i and computed using equation 7.2.

The passage-level attention layer ω_p can be applied in two different places of the passage-level of the general architecture, as shown in figure 8.2. Before the

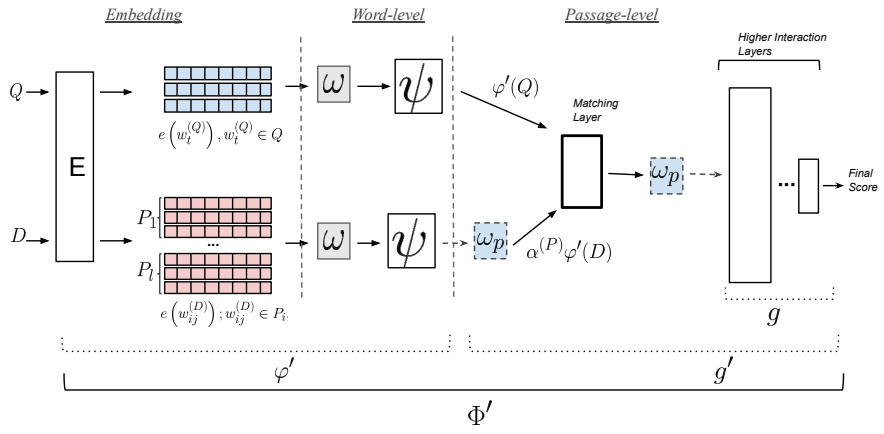


Figure 8.2: Extension of the unified neural matching model of figure 4.1, using attention layers applied in several levels, in order to focus on the most important matching signals at each level.

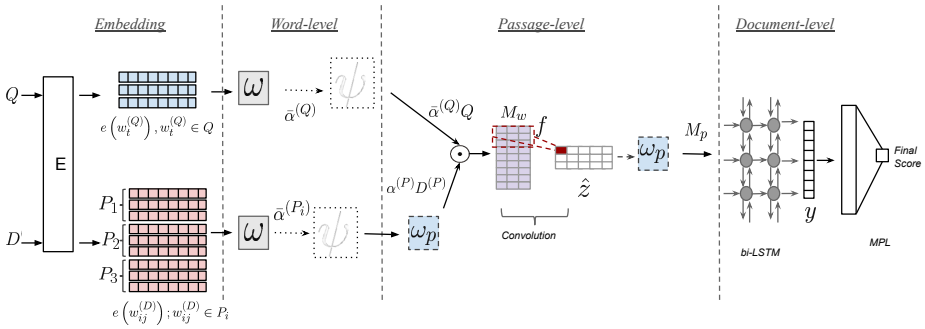


Figure 8.3: Architecture of our AM3 model. The document has three passages, and all the passages and the query have three words. A bi-LSTM layer is used in the document-level.

matching layer, ω_p put attention on the passages based only on the component information independently from the query. After the *matching layer*, ω_p put more attention on some passages based on the lower level interaction features between the query and the document content. The final function g corresponds to the *higher interaction layers* used to compute the matching score.

8.3 AM3: An Attention-based Multi-level Matching Model

We propose an Attention-based Multi-level neural Matching Model (AM3) that exploits three different levels of information granularity: the *word-level*, the *passage-level* and the *document-level*. In the followings, we describe the different levels of our AM3 model accordingly.

8.3.1 Word Level

In the word-level, the aim is to consider words of the input sequences according to their importance. In our model, the function ψ is considered as an *identity* function and represented by dotted lines. At this level, we compute attention weight vectors, $\bar{\alpha}^{(Q)}$ and $\bar{\alpha}^{(P_i)}$ corresponding to all words of the query Q and the passages, respectively, using the attention layer ω defined in equation 7.2.

8.3.2 Passage level

In the word-level, the model is provided with relevance signals corresponding to every word in the different passages of the document but not the passages themselves. At the passage-level, the aim is to better distinguish the most important passages of the whole document. First, we compute a word-word matching matrix $M^{(w)}$ of the query Q and passages of the document D using equation 8.2.

$$M^{(w)} = \left[\bar{\alpha}^{(P_1)} P_1, ..., \bar{\alpha}^{(P_l)} P_l\right] \odot \bar{\alpha}^{(Q)} Q \tag{8.2}$$

where $\odot$ is the cosine similarity.
Then, we compute a relevance vector $\hat{z}_i$ for every passage P_i in D, using a convolutional layer of $k \times |Q|$ dimensional filters, with k is the passages length, applied to the word-word matching matrix $M^{(w)}$, as described in equation 8.3.

$$\hat{z}_i^f \stackrel{def}{=} \delta\left(w^f z_i + b^f\right), f = 1, 2, ..., F \tag{8.3}$$

where z_i corresponds to the convolution window and F is the number of filters, which is also the dimension of the vector $\hat{z}_i$. The parameters w^f and b^f stands for the convolution layer weights and bias, respectively, corresponding to the filter f, and δ is the corresponding activation function.

Once the vector representation $\hat{z}_i$ is computed for every passage, we compute the passage-level attention weight vector $\bar{\alpha}^{(z)} = \left[\alpha_1^{(z)}, ..., \alpha_l^{(z)}\right]$, where $\alpha_i^{(z)}$ is the attention weight of the passage P_i computed using equation 4.8, where the word vector w_t is substituted by the passage vector $\hat{z}_i$. The attention vector $\bar{\alpha}^{(z)}$ enables the matching layers of the document-level to focus on the most important passages. Hence, the passage-level signals to consider in the document-level are given by the matrix $M^{(p)}$ computed as in equation 8.4.

$$M^{(p)} \stackrel{def}{=} \bar{\alpha}^{(z)} \otimes \hat{z} = \left[\alpha_1^{(z)} \hat{z}_1, ..., \alpha_l^{(z)} \hat{z}_l\right] \tag{8.4}$$

where $\otimes$ corresponds to the element-wise product.

8.3.3 Document level

The whole document relevance can be assessed based on the passage-level matching signals. The main idea is that the document can be considered as relevant if there is an important interaction between its different relevant areas. Therefore, at the document-level we construct a document-wide relevance patterns based on signals from the passage-level of the matrix $M^{(p)}$. To this end, we can use one of the two options as a combination layer.

Recurrent Layer (LSTM)

The aim of using a recurrent layer is to process the different passages according to their positions in a document. We assume that a document where the relevant passage appeared at the beginning could be preferred by the user. Hence, the passage-level relevance matrix $M^{(p)}$ is fed to a long short term memory (LSTM) layer [100]. In addition, in order to perform a kind of interaction between the different passages, we use a bidirectional recurrent network, where hidden the states (equation 2.5) $\overrightarrow{h}$ and $\overleftarrow{h}$, of the forward and the backward directions respectively, are concatenated to construct the document-level relevance features as described in equation 8.5.

$$y_{biLSTM} = \left[\overrightarrow{h}, \overleftarrow{h}\right] \tag{8.5}$$

where, if a simple LSTM layer is used, then we consider $y_{LSTM} = \left[\overrightarrow{h}\right]$.

Feedforward Layer (MLP)

The simple feedforward network [270] enables to move the information only in the forward direction of the NN (from input to output), and apply a non-linear transformation at every layer. We fed the matrix $M^{(p)}$ to a multiple layers perceptron (MLP) in order to transform the passage-level relevance signals as described in equation 8.6.

$$y_{MLP} = \delta_F(W^{(F)}M^{(p)} + b^{(F)}) \tag{8.6}$$

where $W^{(F)}$ and $b^{(F)}$ are the feedforward layer parameters, and δ_F is the MLP layer activation function.

Then, a final MLP layer is used to combine the document-level features y and compute the final relevance score sc. As described in equation 8.7.

$$sc = \delta_{sc}(Wy + b) \tag{8.7}$$

where W and b are parameters of the MLP layer, and δ_{sc} is the final layer's activation function. $y \in \{y_{biLSTM}, y_{LSTM}, y_{MLP}\}$ is the final features vector computed in the document level by adopting one of the possible options, namely bidirectional LSTM, simple LSTM or MLP layer.

8.4 Experiments

In this section, we describe the experimental process and the obtained results, and analysis. We will first describe the evaluation setup (section 8.4.1), mainly the dataset and the evaluation protocol, then (section 8.4.3), we will describe the and discuss the obtained results.

8.4.1 Evaluation Setup

We consider the ad-hoc document ranking task and train our AM3 model, and the different evaluated neural models to optimize the rank hinge loss objective

function (equation 7.4). We used the Robust04 news TREC dataset (details about this dataset are given in table 5.5 of section 5.2.3). Robust04 contains 250 queries, hence we performed a 5-fold cross validation, with 150 queries for training, 50 queries for validation and 50 queries for test.

Experiments are conducted using the MatchZoo [5] framework. We have added our AM3 model to MatchZoo. In order to evaluate our passage-based document representation approach, we extended some pre-implemented neural models in this framework with the attention layers ω () and ω_p at several levels, as described in section 8.2.2.
Instead of using all passages of a document, we use passages that are retrieved by the INDRI[2] search engine, with a passages length set[3] to 5 words without overlap.

We have trained our AM3 model using the *Adam* optimizer [271] with a batch size of 100. To avoid overfitting the data, we used a dropout rate with different values in $\{0.0, 0.1, ..., 0.5\}$, in layers of the document-level (section 8.3.3). The filters number F used in equation 8.3 takes values in $\{5, 10, ..., 100\}$. Our model as well as the neural baselines use the pre-trained Glove[4] word embeddings with embedding size 300. All the models are trained over 600 epochs, then evaluated according to their performances in the last epoch.
Concerning the hyper-parameters of the different baselines, we have considered the models configuration corresponding to the best results published in the corresponding paper.

8.4.2 Models Configuration

We have extended several neural baselines described in section 5.4, using different attention layers as described in section 8.2.2.

- *Original* refers to the original model without any modification.
- $\omega.P$ refers to the use of the attention layer ω_p (equation 8.1), at the passage-level, to compute attention weights for the different passages.
- $\omega.Q$ refers to the attention weights computed by the layer ω (equation 7.2) at the word-level of the query input.
- $\omega.D$ refers to the attention weights computed, by layer ω (equation 7.2), at the word-level of the different passages of the input document.

For the DUET model [179], we considered different cases. In fact, this model is composed of two parallel convolutional models, the *local* and the *distributed*. We refereed for each of them with the labels *local* and *distrib* respectively. The *localDistrib* label refers to the application of the attention layer, $\omega.Q$, $\omega.P$ and/or $\omega.D$ in the corresponding level at both *local* and *distributed* parts of the DUET models.

We have also considered different configurations of our AM3 model for the document-level and the passage-level processing. The evaluated configurations are refereed to using the following labels:

[2] http://lemurproject.org/indri.php
[3] The passage length took values from $\{5, 10, ..., 100\}$ then we chose the value that gave the best results.
[4] http://nlp.stanford.edu/data/glove.840B.300d.zip

Document Representation	Models	Performance						
	Model	MAP	P@1	P@5	P@10	nDCG@1	nDCG@5	nDCG@10
Document Content	BM25	**0.237**	**0.528**	**0.468**	**0.424**	**0.459**	**0.444**	**0.428**
	ARC-II	0.063	0.044	0.057	0.064	0.062	0.041	0.055
	DUET	0.117	0.237	0.170	0.155	0.218	0.170	0.160
	MatchPyramid	0.118	0.169	0.162	0.163	0.150	0.146	0.150
	MV-LSTM	0.102	0.116	0.114	0.112	0.106	0.105	0.106
	DRMM	0.155	0.341	0.255	0.247	0.317	0.250	0.248
Passages	BM25	0.220	0.320	0.257	0.232	0.252	0.243	0.234
	ARC-II	0.149	0.136	0.160	0.156	0.128	0.149	0.157
	DUET	0.179	0.228	0.218	0.200	0.215	0.209	0.211
	MatchPyramid	0.130	0.120	0.126	0.121	0.115	0.116	0.122
	MV-LSTM	0.164	0.172	0.181	0.166	0.153	0.170	0.170
	DRMM	**0.336**	**0.504**	**0.410**	**0.356**	**0.467**	**0.402**	**0.375**

Table 8.1: Performances evaluation of several baselines, in the Robust04 dataset, using the document content compared to the passage-based content.

- *FL* refers to the use of a feedforward layer to combine the passage-level features (section 8.3.3).
- *LSTM* refers to the use of a simple LSTM layer to combine the passage-level features. In this case, the document-level features computed in equation 8.5 are $y = \left[\overrightarrow{h}\right]$.
- *bi-LSTM* refers to the use of a bidirectional LSTM layer to combine the passage level features in equation 8.5.

Note that, in case the $\omega.P$ label is not used, the passage-level features tensor of equation 8.4 is $M^{(p)} = \hat{z}$.

8.4.3 Results and Discussion

We considered the performance evaluation in terms of precision (P) and normalized discounted cumulative gain (nDCG), at ranks 1, 5 and 10 in addition to the mean average precision (MAP). The objective is to answer several research questions, as follows.

***RQ1:* Does the passage-based document representation approach correctly capture relevant documents?**

First, we evaluate the impact of using the passage-based document representation, compared to the use of the whole document content to run existing state-of-the-art models, and show how it affects the models performances. We used some neural models that are implemented in MatchZoo, in addition to the BM25[5] classical model implemented in INDRI.
Table 8.1 shows the performances of the different evaluated models. In this table, note that the BM25 model does not benefit from the passage-based document representation. Indeed, the new document content (based on the concatenated

[5]While using the BM25 model, we concatenated the different passages, of the same document, that are retrieved by INDRI, to construct the new document content.

Models	Configurations	MAP	P@1	P@5	P@10	nDCG@1	nDCG@5	nDCG@10
ARC-II	Original	0.149	0.136	0.160	0.156	0.128	0.150	0.157
	$\omega.P$	0.132	0.124	0.146	0.139	0.105	0.132	0.137
	$\omega.P$+$\omega.Q$	0.137	0.152	0.143	0.142	0.144	0.141	0.147
	$\omega.P$+$\omega.D$	**0.175**	**0.172**	**0.216**	**0.204**	**0.168**	**0.200**	**0.206**
	$\omega.P$+$\omega.D$+$\omega.Q$	0.153	0.168	0.149	0.140	0.157	0.146	0.149
DUET	Original	0.179	0.228	0.218	0.200	0.215	0.209	0.211
	$\omega.P$+local	0.175	0.268	0.212	0.192	0.247	0.209	0.209
	$\omega.P$+distrib	0.157	0.196	0.187	0.169	0.177	0.177	0.180
	$\omega.P$+localDistrib	0.191	0.280	0.230	0.211	0.251	0.232	0.230
	$\omega.P$+local+$\omega.Q$	0.176	0.240	0.216	0.198	0.221	0.210	0.212
	$\omega.P$+local+$\omega.D$	**0.235**	0.324	**0.281**	**0.254**	**0.308**	**0.284**	**0.281**
	$\omega.P$+local+$\omega.D$+$\omega.Q$	0.226	0.312	0.270	0.250	0.288	0.271	0.270
	$\omega.P$+distrib+$\omega.Q$	0.153	0.188	0.165	0.158	0.169	0.165	0.168
	$\omega.P$+distrib+$\omega.D$	0.178	0.200	0.208	0.204	0.189	0.202	0.210
	$\omega.P$+distrib+$\omega.D$++$\omega.Q$	0.176	0.220	0.192	0.186	0.212	0.191	0.199
	$\omega.P$+localDistrib+$\omega.Q$	0.176	0.240	0.216	0.198	0.221	0.210	0.212
	$\omega.P$+localDistrib+$\omega.D$	0.230	0.304	0.273	0.250	0.283	0.274	0.273
	$\omega.P$+localDistrib+$\omega.D$+$\omega.Q$	0.226	**0.336**	0.279	**0.254**	0.307	0.275	0.275
MatchPyramid	Original	0.130	0.120	0.126	0.121	0.115	0.116	0.122
	$\omega.P$	0.186	0.232	0.194	0.186	0.197	0.197	0.203
	$\omega.P$+$\omega.Q$	**0.236**	**0.320**	**0.278**	**0.240**	**0.299**	**0.282**	**0.269**
	$\omega.P$+$\omega.D$	0.156	0.168	0.158	0.141	0.151	0.150	0.151
	$\omega.P$+$\omega.D$+$\omega.Q$	0.135	0.244	0.174	0.146	0.193	0.172	0.160
MV-LSTM	Original	0.164	0.172	0.181	0.166	0.153	0.170	0.170
	$\omega.P$	0.177	**0.228**	0.196	0.183	**0.215**	**0.192**	0.191
	$\omega.P$+$\omega.Q$	0.179	0.176	0.191	0.190	0.171	0.182	**0.193**
	$\omega.P$+$\omega.D$	0.168	0.152	0.186	0.178	0.149	0.173	0.178
	$\omega.P$+$\omega.D$+$\omega.Q$	**0.183**	0.164	**0.204**	**0.193**	0.156	0.188	0.194

Table 8.2: Performances evaluation of several state-of-the-art models extended with attention layers at different levels.

passages) contains words with less frequencies than the original document content. Hence, query words frequencies are also reduced, and this seems to be the reason why the BM25 performances decreased.

While considering the neural models, the document content corresponds to only the first 200 words of every document, which corresponds to the maximum document length supported by the different evaluated neural models in MatchZoo in this experiments. However, as shown in section 8.1, relevance signals are located far away from the first 200 words. We notice that the passage-based document representation has a positive impact on the different neural models performances, compared to the use of a cut document content. For instance, performances in terms of MAP, of the models ARC-II, DUET and the MV-LSTM are better, with respectively more than 136%, 52% and 60% when using the passage-based representations than when using the document content.

Moreover, results of the DRMM model are better than the BM25 using the passage-based representations, in terms of MAP and nDCG@1. These results support our hypothesis concerning the relevance distribution over the long documents, and the information loss engendered by the cutting off of the document content. Indeed, the passage-based document representation makes it possible to reduce the size of the document, and thus make it bearable by the neural models. In addition, the passages selection process (section 8.2.1) constructs a document content that necessarily contains the query words, which help to better assess the document relevance.

***RQ2:* What is the impact of the multi-level attention weights in state-of-the-art models?**

In addition to the passage-based document representation, we investigate how the attention weights computed at different levels (figure 8.2) impact the performances of existing neural matching models. Table 8.2 shows the performance results of several extended neural baselines with the attention layers ω and ω_p applied at different levels within different configurations. We used different labels in order to refer to each of them as described in section 8.4.2.

In this table, we can notice that at least one configuration of the extended models performs better than its original counterpart. The best performances of every model are highlighted with bold characters. Best results are obtained by the DUET model using the $\omega.P$+local+$\omega.D$ extension. The latter improves the performance with more than 31% in terms of MAP and more than 28% in terms of nDCG@5, compared to the corresponding original model. Note that for this analysis, we do not consider the DRMM model [178]. In fact, this model uses a histogram function for mapping the word-level matching signals into lower-sized vectors, where every vector corresponds to the matching histogram of one query word with all the document.

Most of the model extensions using the attention layer ω_p at the passage-level combined with the attention layer applied at the word-level of the document ($\omega.P + \omega.D$) or of the query ($\omega.P + \omega.Q$) perform better than the original versions as well as the extensions using the attention layer at the passage level only, the word level only or both of them at once ($\omega.P + \omega.D + \omega.Q$). In fact, the word-level weights enable to identify the core information of every passage, and the passage-level weights consider the importance differences among the different passages. For instance, in case of ARC-II, the configuration $\omega.P$+$\omega.D$ outperforms all the other configurations, improving the performance of the original model with more than 35% in terms of P@5. For the MatchPyramid model, the $\omega.P$+$\omega.Q$ extension outperforms all the other models. Such that the results are more than 166% better than those of the original MatchPyramid model in terms of P@1. In fact, the attention weights computed for the different words in the query inform the model structure to focus on the passages having the most important words of the query.

Moreover, the $\omega.P$+local+$\omega.D$ extension of the DUET model performs better results than all the other extensions of this model, except for the P@1 where the $\omega.P$+localDistrib+$\omega.D$+$\omega.Q$ extension outperforms the original model with more than 47% in terms of P@1 compared to the 42% improvements with the $\omega.P$+local+$\omega.D$ extension. In fact, $\omega.D$ enables to identify the most important words of the different passages, in addition to $\omega.P$ enabling to focus the interaction process of both the *local* and *distributed* parallel models of DUET on the most relevant passages.

The conclusion to be drawn from these results is that when the attention is put on the word-level, attention weights must be computed exclusively for words of only one of the two inputs at a time, the query input or the document input, resulting in an asymmetric configuration of the model. Note that, for most

Model Configuration		MAP	P@1	P@5	P@10	nDCG@1	nDCG@5	nDCG@10
Feedforward Layer	AM3.FL	0.218	0.316	0.251	0.235	0.283	0.249	0.251
	AM3.FL+$\omega.Q$	0.208	0.304	0.258	0.231	0.257	0.252	0.249
	AM3.FL+$\omega.D$	0.185	0.212	0.225	0.211	0.171	0.207	0.211
	AM3.FL+$\omega.Q$+$\omega.D$	0.199	0.276	0.241	0.222	0.252	0.231	0.235
	AM3.FL+$\omega.P$	0.241	0.348	0.303	0.252	0.321	0.296	0.279
	AM3.FL+$\omega.P$+$\omega.Q$	0.213	0.308	0.254	0.233	0.272	0.249	0.249
	AM3.FL+$\omega.P$+$\omega.D$	0.232	0.304	0.279	0.264	0.272	0.266	0.274
	AM3.FL+$\omega.P$+$\omega.Q$+$\omega.D$	0.207	0.288	0.254	0.226	0.249	0.244	0.245
Recurrent Layer	AM3.LSTM	0.210	0.316	0.281	0.244	0.244	0.266	0.259
	AM3.LSTM+$\omega.Q$	0.220	0.328	0.293	0.264	0.287	0.289	0.272
	AM3.LSTM+$\omega.D$	0.232	0.336	0.302	0.271	0.316	0.296	0.293
	AM3.LSTM+$\omega.Q$+$\omega.D$	0.184	0.240	0.246	0.210	0.220	0.222	0.226
	AM3.LSTM+$\omega.P$	0.228	0.328	0.274	0.25	0.307	0.265	0.264
	AM3.LSTM+$\omega.P$+$\omega.Q$	0.226	0.348	0.296	0.261	0.295	0.287	0.276
	AM3.LSTM+$\omega.P$+$\omega.D$	0.217	0.344	0.282	0.250	0.308	0.284	0.272
	AM3.LSTM+$\omega.P$+$\omega.Q$+$\omega.D$	0.181	0.264	0.234	0.207	0.225	0.222	0.214
	AM3.biLSTM	0.209	0.28	0.245	0.225	0.257	0.24	0.237
	AM3.biLSTM+$\omega.Q$	0.220	0.328	0.292	0.260	0.288	0.283	0.277
	AM3.biLSTM+$\omega.D$	0.199	0.24	0.235	0.218	0.197	0.228	0.231
	AM3.biLSTM+$\omega.Q$+$\omega.D$	0.161	0.140	0.164	0.169	0.148	0.165	0.167
	AM3.biLSTM+$\omega.P$	0.202	0.316	0.238	0.229	0.268	0.232	0.237
	AM3.biLSTM+$\omega.P$+$\omega.Q$	**0.245**	**0.384**	**0.327**	**0.290**	**0.343**	**0.315**	**0.304**
	AM3.biLSTM+$\omega.P$+$\omega.D$	0.228	0.292	0.260	0.238	0.268	0.255	0.256
	AM3.biLSTM+$\omega.P$+$\omega.Q$+$\omega.D$	0.216	0.316	0.254	0.222	0.295	0.251	0.244

Table 8.3: Performances evaluation of the AM3 model using different configurations, in the Robust04 dataset.

the evaluated models, results of one of the asymmetric configurations, namely the $\omega.P$+$\omega.D$ or the $\omega.P$+$\omega.Q$, are better than the corresponding symmetric configuration. $\omega.P$+$\omega.Q$+$\omega.D$. This is due to the *asymmetric* nature of the query-document matching. As it is discussed in a similar task (question-answer matching) in chapter 7.

***RQ3:* What are the performances of the different configurations of the AM3 model?**

To answer this question, we used different configurations of our AM3 model. The objective is to evaluate the impact of putting the attention at different levels (word-level and/or passage-level). More specifically, we evaluated the impact of the attention weights computed for the different passages, referred to with label $\omega.P$; and the impact of the attention weights at the word-level of the query, referred to with label $\omega.Q$, and of the document, referred to with label $\omega.D$. Furthermore, we evaluate the impact of the recurrent layers (LSTM vs bi-LSTM), used in the document-level to combine the passage-level features, compared to the feedforward layer (FL). Table 8.3 shows performances of the different configurations accordingly.

In this table, the underlined values show the best performance over the configurations using the feedforward layer, and the best ones over those using the recurrent layers, LSTM or biLSTM. The best performances over all the different configurations are highlighted with **bold** characters.

Note that the attention at the passage-level (label $\omega.P$) has a clear impact in performances. Results of most of the model variants where the attention layer is applied at the passage-level, are better than those of their counterparts where this layer is not used. Moreover, the model variant AM3.biLSTM+$\omega.P$+$\omega.Q$ gives the best results of the AM3 model. Indeed, the attention layer applied at the word level of the query (label $\omega.Q$) helped improving better the results,

when combined with the attention weights computed to the different passages (label $\omega.P$). However, when the word-level attention weights are computed for the document words (label $\omega.D$), results are not as good as those of the configurations having label $\omega.Q$. Hence, when the document-level is provided with the weighted passage representations as well as signals about the most important query words, results are better. For the second time, the advantage of an *asymmetric* configuration in performing the asymmetric matching task (query-document) is observed, as was the case in results of table 8.4.3.

Concerning the document-level interactions using the recurrent layers or the feedforward layer, we notice that the best performances reached by the variants using a recurrent layer, in particular the AM3.biLSTM+$\omega.P$+$\omega.Q$ model outperforms the best results of the model configuration using a simple feedforward layer, namely the AM3.FL+$\omega.P$ model. Indeed, the recurrent layer takes into account the passages sequencing in the memory cells [100]. Hence, the recurrent layer provides more relevant signals than those provided by the simple feedfoward layer to the final matching layer of our model.

8.5 Conclusion

In this chapter, we propose a passage-based document representation method for ad-hoc ranking of documents. Unlike the common solution that consists on cutting off the exceeded text of a document to make it bearable by a neural model.

We first, show that several matching signals are located far away in the document (after the first 200 words), and combining different pre-ranked passages of a document enables to get better results. We then suggest to represent documents as sets of passages that are likely to be relevant. Based on these representations, we extended several neural matching models from the state-of-the-art, using attention layers. We noticed the improvements of all the extended models' results using the attention weights at different levels. Note that, the asymmetric configurations where the attention is put on the passages and their words (configurations $\omega.P$+$\omega.D$) or words of the query (configurations $\omega.P$+$\omega.Q$) have better results than those of the symmetric configurations ($\omega.P$+$\omega.Q$+$\omega.D$) where attention is put on both document and query words and the different passages. Such that, when the passages are used instead of the whole document content, results are improved with more than 136% in terms of MAP. Besides, while using the extended versions with the attention layers at different levels, the improvements were up to 166% in terms of P@1. Which support our hypothesis about the multi-level relevance granularity in a document.

Furthermore, we proposed a multi-level matching model. Results confirmed for a second time the advantage of an asymmetric configuration of the multi-level matching, to handle the query-document matching.

Part V

Overall Conclusion

CONCLUSION

Contributions Overview

The work presented in this thesis is part of the general context of information retrieval, and it focuses specifically in deep learning models.

We were particularly interested in IR models that exploit distributed representations of words, and those using neural models to learn matching features. We have focused the state-of-the-art part on these two lines of research, and we have devoted two chapters to the description of the different models proposed in this framework. Specifically, chapter 3 focuses on models exploiting distributed representations of words and sequences to improve classical IR models, such as the enhanced language model with word embeddings [159] and the MBM25EMB model [188] that extended the BM25 using word embeddings. Chapter 4 focuses on models using neural networks to, first capture distributional semantics in the corpus of different documents and construct latent representations, then learn the matching function comparing these representations to better rank relevant results.

In this context, we addressed four main questions:

1. While performing the document-query matching based on the semantic matching using word embeddings, most of the proposed models [183, 14] handle the query words regardless of their occurrence or not in the documents. We assume that query words contribute to the matching process in different ways, depending on their presence or absence in every document. Hence, we aim to answer this question: *How do the different terms of the query contribute to the relevance assessment of different documents, according to their presence/absence in every document?*

   In order to answer this question, we analyzed the the impact of presence/absence of query words in the matching process using word embedding [259]. To do so, we compared three different matching strategies, where every word in a document is weighted using a classical IR model, and then compared to the different query words, using the cosine distance between their embedded vectors. Each of the three strategies focuses on a specific aspect:

   - All the words of the query are equi-important. Hence, we match them with all document terms in the same way.
   - The presence/absence of a query word in a document maters. Hence, we match the query words with those of the document accordingly.

- The presence/absence of a query word in a document is an important aspect, and the exact matches are of different importance than the semantic matches. Hence, we compare the query terms with the document terms, based on their presence in every document, while highlighting their similarities based on exact/semantic matching.

The experimental results show that considering query terms that are absent in the document, differently from those that are not, improves the performances of traditional models using embedded word vectors. More specifically, in datasets where the major of relevant documents do not contain any/most of exact matches with the query words, the proposed matching strategies performed better results.

2. In a second step, we focused our work on text matching using neural models. Indeed, the text matching applications include several tasks that are of various natures. However, most existing neural matching models [25, 236, 231, 110] handle the input sequences in the same way, and their representations are computed using the same[6] function, whatever is the nature of the matching task. In this case, the question is: *What are the main distinctions between the different matching tasks? and how a neural matching model can deal with the different inputs accordingly?*

   To answer this, we first highlighted the *asymmetry* aspect of the matching task involving input sequences of different natures, such as question-answer and query-document matching, then proposed a neural matching architecture that enables to handle this aspect [252]. More specifically, this architecture uses attention layers, such that the model's configuration correspond to the nature of the task. The experimental results, in question-answer matching as an *asymmetric* task and paraphrase identification as a *symmetric* task, have shown a great performance of the adaptable architecture, while extending several state-of-the-art models. We concluded that for an *asymmetric* matching task, it is better to have an *asymmetric* model. While for *symmetric* tasks, the attention layers enable to improve the performances of some models, but the differences are not significant.

3. Based on the *asymmetry* sensitive approach we propose [252], we analyzed some word-level features in neural matching models. More specifically, the impact of *attention* and *position* features on text matching. Indeed, the recently proposed attention-models [31] put more attention on the most important words in a given sequence, while the position-based models [3] leverage the position information of every word in a sequence. The question that we aim to answer is: *How the attention-based and position-based approaches can help improving the performance of neural models for text matching?*

   In order to answer this question, we extended the positional model MV-LSTM [3] using attention layers at the word-level, and evaluated different configurations [272], according to the nature of the matching task, symmetric or asymmetric [252]. Experimental results have shown that the

[6] We consider two representation layers having similar architectures as same functions.

combining the attention weights with the position information, in an asymmetric configuration, improved results of the original model [3], based on only position features, with more than 3.8% and 7% in terms of accuracy and MAP, respectively.

4. Finally, while using neural networks for ad-hoc ranking of long documents, most neural models struggle dealing with the document length, and the common solution is to set a maximum document length [21]. According to several studies that have already been performed [27], this solution leads to an information loss. In another hand, the passage-based models [29, 30] have already shown great results in ranking long documents. In this case, the question that raises is: *How the passage-retrieval techniques can help neural models to better rank documents?*

   In order to solve this problem, in a current work, we consider two assumptions [252]: (1) every document can be represented as a set of passages, where the different passages are not of the same importance; (2) there are three different relevance granularities in a document, namely the word-level, the passage-level and the document-level. To handle these aspects, we proceed as follows:

   - We proposed a multi-level architecture to extend different state-of-the-art matching models, using attention layers at different levels. The documents are represented as sets of passages, and attention weights are computed for words of the query and the passages, and for the different passages. The aim is to focus the matching function on most important words of the query and the document, as well as the most relevant passages of this one.
   - We extended several state-of-the-art neural models, using the proposed architecture. Such that for every extended model, we compared different configurations according to the attention layers, in order to show the advantages of considering the attention weights at different levels.
   - Furthermore, we proposed our attention-based multi-level matching model (AM3). We considered different configurations: at the word-level, we compute attention weights for words of the query and the document. At the passage-level, we used a convolution layer applied to a word-word matching matrix, using a set of filters to compute the embedded vector of every passage, then compute attention weights for the different passages. Finally, in the document-level, we compared different combination layers, in order to compute the relevance score of the whole document.

   The experimental results show that the passage-based document representations have helped improving performances of several state-of-the-art neural matching models, with more than 136% in terms of MAP compared to those using the limited document content. Besides, the attention put on the different passages have improved results of the extended models are up to 166% in terms of P@1, when using an asymmetric configurations. Finally, the results of our AM3 model, using an asymmetric configura-

tion too, are better than the best results corresponding to the extended models.

Perspectives and Future Work

The work reported in this thesis could be extended in several directions.
First of all, it would be interesting to provide additional validations of our models using larger datasets. For instance, the SQUAD dataset [273] which contains more than 100k questions can be used in order to evaluate our contributions in the question answering tasks. In order to evaluate the document ranking models, the LETOR 4.0 dataset [213] provides a large set of evaluated documents and computed features for LTR models. This dataset uses the GOV2 web page collection (25M pages) and two query sets from the Million Query track of TREC 2007 and TREC 2008. Another dataset is provided recently in the TREC 2019 Deep Learning Track[7]. Namely MS-MARCO [274], this dataset is intended to evaluate deep learning models for text matching. It includes six different parts, among which parts dedicated to document ranking using labeled passages, which can be used in order to validate the results of the contributions in chapters 6 and 8, as well as parts for question answering tasks that can be used to evaluate the contributions in chapter 7.

In all of the different contributions, we only considered the traditional word embeddings, such as Word2Vec [2] and GloVe [9]. However, recent works [200, 201] consider *contextual* word embedding representations, such as BERT [104], where every occurrence of a word in the corpus can be represented differently. The advantages of this kind of embeddings is that the different meanings of a same word are not considered to be the same. Hence, the results of the different matching strategies shown in chapter 6 could be improved by considering a contextual word embeddings.
Besides, the neural matching architectures proposed in chapters 7 and 8 can be extended in order to include the pre-trained contextual word embeddings, as additional features of the different input sequences, in order to handle the different word contexts accordingly. To do this, the contextual embedding layer can be used instead of the regular embedding layer. Then, compute attention weights for the different context-based embedded vectors. The aim is to enable the model, trained in a learning to rank setup, to focus more on the context of a word in the relevant document, and thus learn to distinguish between the word used in a relevant context and the same word used in an irrelevant context.

Another important perspective is related to the neural matching architecture proposed in section 7.2. Indeed, the model proposed in this section aims to address the *asymmetry* aspect of some text matching tasks, such as question-answer and similar tasks. Through the different analysis of the results of this model, we made two main conclusions: (1) when the matching task is *asymmetric* the architecture's configuration that performs better results corresponds to one of the *asymmetric* configurations, (Q) or (A), the same is true when it is a *symmetric* task on which the *symmetric* architectures are either the (Q+A) configuration or the model's *original* configuration; (2) there is a clear correlation between the the questions type (e.g. *when*, *who*...) and the architecture

[7] https://microsoft.github.io/TREC-2019-Deep-Learning/

that performed best results of the different evaluated models. Hence, we estimated the performance of the Oracle version corresponding to every extended model and we figured out that its results could reach state-of-the-art results. To achieve this, constructing a model architecture that is able to automatically adapt its configuration (Q), (A) or (Q+A), according to the question's type as well as the nature of the matching task being addressed, is an important direction to explore. This perspective can also be extended for query-document matching, where the documents can be represented as sets of passages.

BIBLIOGRAPHY

[1] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[3] Shengxian Wan, Yanyan Lan, Jiafeng Guo, Jun Xu, Liang Pang, and Xueqi Cheng. A deep architecture for semantic matching with multiple positional sentence representations. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[4] Yi Yang, Wen-tau Yih, and Christopher Meek. WikiQA: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[5] Yixing Fan, Liang Pang, JianPeng Hou, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. Matchzoo: A toolkit for deep text matching. *arXiv preprint arXiv:1707.07270*, 2017.

[6] Gerard Salton. Information storage and retrieval. reports on analysis, search, and iterative retrieval. 1968.

[7] Mustapha Baziz, Mohand Boughanem, Nathalie Aussenac-Gilles, and Claude Chrisment. Semantic cores for representing documents in ir. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, SAC '05, pages 1011–1017, New York, NY, USA, 2005. ACM.

[8] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

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

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[11] Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W. Bruce Croft, and Xueqi Cheng. A deep look into neural ranking models for information retrieval. *Information Processing and Management*, page 102067, 2019.

[12] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3294–3302. Curran Associates, Inc., 2015.

[13] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International Conference on Machine Learning*, pages 957–966, 2015.

[14] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce . Semantic matching by non-linear word transportation for information retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 701–710. ACM, 2016.

[15] Yanshan Wang, Sijia Liu, Naveed Afzal, Majid Rastegar-Mojarad, Liwei Wang, Feichen Shen, Paul Kingsbury, and Hongfang Liu. A comparison of word embeddings for the biomedical natural language processing. *Journal of biomedical informatics*, 87:12–20, 2018.

[16] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338. ACM, 2013.

[17] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems*, pages 2042–2050, 2014.

[18] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[19] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. Text matching as image recognition. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[20] Gaurav Singh Tomar, Thyago Duque, Oscar Täckström, Jakob Uszkoreit, and Dipanjan Das. Neural paraphrase identification of questions with noisy pretraining. pages 142–147, 2017.

[21] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Jingfang Xu, and Xueqi Cheng. Deeprank: A new deep architecture for relevance ranking in information retrieval. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 257–266, New York, NY, USA, 2017. ACM.

[22] Yongxing Peng and Bo Liu. Attention-based neural network for short-text question answering. In *Proceedings of the 2018 2Nd International Conference on Deep Learning Technologies*, ICDLT '18, pages 21–26, New York, NY, USA, 2018. ACM.

[23] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*, pages 737–744, 1994.

[24] Yang Song, Qinmin Vivian Hu, and Liang He. P-cnn: Enhancing text matching with positional convolutional neural network. *Knowledge-Based Systems*, 169:67–79, 2019.

[25] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

[26] Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255, 2016.

[27] Yixing Fan, Jiafeng Guo, Yanyan Lan, Jun Xu, Chengxiang Zhai, and Xueqi Cheng. Modeling diverse relevance patterns in ad-hoc retrieval. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '18, pages 375–384, New York, NY, USA, 2018. ACM.

[28] Xiaoyong Liu and W Bruce Croft. Passage retrieval based on language models. In *11th international conf. CIKM'02*, 2002.

[29] Mengqiu Wang and Luo Si. Discriminative probabilistic models for passage based retrieval. In *31st international ACM SIGIR'08 conf.*, 2008.

[30] Yuanhua Lv and ChengXiang Zhai. Positional language models for information retrieval. In *32nd international ACM SIGIR'09 conf.*, 2009.

[31] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[32] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103, 2010.

[33] Hans Peter Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, 1(4):309–317, 1957.

[34] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[35] Chengxiang Zhai and John Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, CIKM '01, pages 403–410, New York, NY, USA, 2001. ACM.

[36] Susan T Dumais. Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments, & Computers*, 23(2):229–236, 1991.

[37] Suzanne Briet. *Qu'est-ce que la documentation?*, volume 1. Éditions documentaires, industrielles et techniques, 1951.

[38] Michael K Buckland. What is a "document"? *Journal of the American society for information science*, 48(9):804–809, 1997.

[39] David M Levy. Fixed or fluid?: document stability and new media. In *Proceedings of the 1994 ACM European conference on Hypermedia technology*, pages 24–31. ACM, 1994.

[40] Bernard J Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic. Real life information retrieval: A study of user queries on the web. In *Acm sigir forum*, volume 32, pages 5–17. ACM, 1998.

[41] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. In *ACm SIGIR Forum*, volume 33, pages 6–12. ACM, 1999.

[42] Carlos A Cuadra. *Experimental Studies of Relevance Judgments. Final Report [by Carlos A. Cuadra and Others]*. System Development Corporation, 1967.

[43] William S Cooper. A definition of relevance for information retrieval. *Information storage and retrieval*, 7(1):19–37, 1971.

[44] Stefano Mizzaro. Relevance: The whole history. *Journal of the American society for information science*, 48(9):810–832, 1997.

[45] Jiaxin Mao, Yiqun Liu, Ke Zhou, Jian-Yun Nie, Jingtao Song, Min Zhang, Shaoping Ma, Jiashen Sun, and Hengliang Luo. When does relevance mean usefulness and user satisfaction in web search? In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 463–472, New York, NY, USA, 2016. ACM.

[46] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[47] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[48] David D Lewis. Text representation for intelligent text retrieval: A classification-oriented view. *Text-based intelligent systems: current research and practice in information extraction and retrieval*, pages 179–197, 1992.

[49] Donald Metzler and W Bruce . Combining the language model and inference network approaches to retrieval. *Information processing & management*, 40(5):735–750, 2004.

[50] Matthew W. Bilotti, Paul Ogilvie, Jamie Callan, and Eric Nyberg. Structured retrieval for question answering. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 351–358, New York, NY, USA, 2007. ACM.

[51] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[52] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, 2013.

[53] Thomas K Landauer and Susan T Dumais. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 104(2):211, 1997.

[54] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[55] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[56] Omer Levy and Yoav Goldberg. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the eighteenth conference on computational natural language learning*, pages 171–180, 2014.

[57] Stephen E Robertson and Steve Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *17th international ACM SIGIR'94 conf.* Springer-Verlag New York, Inc., 1994.

[58] Donald Metzler. Generalized inverse document frequency. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 399–408, New York, NY, USA, 2008. ACM.

[59] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975.

[60] Jay Michael Ponte and W Bruce Croft. *A language modeling approach to information retrieval*. PhD thesis, University of Massachusetts at Amherst, 1998.

[61] Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[62] Charu C Aggarwal and ChengXiang Zhai. *Mining text data*. Springer Science & Business Media, 2012.

[63] Ye Zhang, Md Mustafizur Rahman, Alex Braylan, Brandon Dang, Heng-Lu Chang, Henna Kim, Quinten McNamara, Aaron Angert, Edward Banner, Vivek Khetan, et al. Neural information retrieval: A literature review. *arXiv preprint arXiv:1611.06792*, 2016.

[64] Bernard J Jansen and Amanda Spink. Analysis of document viewing patterns of web search engine users. In *Web mining: Applications and techniques*, pages 339–354. IGI Global, 2005.

[65] James W Perry, Kent Allen, and Madeline M Berry. Machine literature searching x. machine language; factors underlying its design and development. *American Documentation (pre-1986)*, 6(4):242, 1955.

[66] Charles L.A. Clarke, Maheedhar Kolla, Gordon V. Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 659–666, New York, NY, USA, 2008. ACM.

[67] William Goffman. A searching procedure for information retrieval. *Information Storage and Retrieval*, 2(2):73–78, 1964.

[68] Andrew Turpin and Falk Scholer. User performance versus precision measures for simple search tasks. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 11–18, New York, NY, USA, 2006. ACM.

[69] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[70] Patrick Paroubek, Stéphane Chaudiron, and Lynette Hirschman. Principles of evaluation in natural language processing. *Traitement Automatique des Langues*, 48(1):7–31, 2007.

[71] Susan T Dumais, George W Furnas, Thomas K Landauer, Scott Deerwester, and Richard Harshman. Using latent semantic analysis to improve access to textual information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 281–285. Acm, 1988.

[72] Julio Gonzalo, Felisa Verdejo, Irina Chugur, and Juan Cigarran. Indexing with wordnet synsets can improve text retrieval. *arXiv preprint cmp-lg/9808002*, 1998.

[73] Mark Sanderson. Word sense disambiguation and information retrieval. In *SIGIR'94*, pages 142–151. Springer, 1994.

[74] Avi Arampatzis and Jaap Kamps. A study of query length. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 811–812. Citeseer, 2008.

[75] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[76] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[77] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[78] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[79] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[80] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[81] Mohand Boughanem, Taoufiq Dkaki, Josiane Mothe, and C Soule-Dupuy. Mercure at trec7. In *TREC*, volume 1998, pages 355–360. Citeseer, 1998.

[82] John Nickolls, Ian Buck, and Michael Garland. Scalable parallel programming. In *2008 IEEE Hot Chips 20 Symposium (HCS)*, pages 40–53. IEEE, 2008.

[83] Jerome Y Lettvin, Humberto R Maturana, Warren S McCulloch, and Walter H Pitts. What the frog's eye tells the frog's brain. *Proceedings of the IRE*, 47(11):1940–1951, 1959.

[84] Giovanni Alcantara. Empirical analysis of non-linear activation functions for deep neural networks in classification tasks. *arXiv preprint arXiv:1710.11272*, 2017.

[85] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. Cambridge MA: MIT press, 1969.

[86] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[87] Richard P Lippmann. Pattern classification using neural networks. *IEEE communications magazine*, 27(11):47–50, 1989.

[88] Donald F Specht. A general regression neural network. *IEEE transactions on neural networks*, 2(6):568–576, 1991.

[89] John S Denker, WR Gardner, Hans Peter Graf, Donnie Henderson, Richard E Howard, W Hubbard, Lawrence D Jackel, Henry S Baird, and Isabelle Guyon. Neural network recognizer for hand-written zip code digits. In *Advances in neural information processing systems*, pages 323–331, 1989.

[90] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[91] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[92] Michael Egmont-Petersen, Dick de Ridder, and Heinz Handels. Image processing with neural networks—a review. *Pattern recognition*, 35(10):2279–2301, 2002.

[93] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 565–571. IEEE, 2016.

[94] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[95] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

[96] Kazuya Kawakami. *Supervised sequence labelling with recurrent neural networks*. PhD thesis, Ph. D. thesis, Technical University of Munich, 2008.

[97] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[98] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. Learning text similarity with siamese recurrent networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 148–157, 2016.

[99] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[100] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[101] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005.

[102] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[103] Alec Radford, Jeffrey Wu, Dario Amodei, Daniela Amodei, Jack Clark, Miles Brundage, and Ilya Sutskever. Better language models and their implications. *OpenAI, 2018b. URL https://openai. com/blog/better-language-models*, 2019.

[104] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[105] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[106] Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S. Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 541–550, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[107] Nikhil Rasiwasia and Nuno Vasconcelos. Scene classification with low-dimensional semantic spaces and weak supervision. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–6. IEEE, 2008.

[108] Shangxuan Tian, Shijian Lu, and Chongshou Li. Wetext: Scene text detection under weak supervision. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1492–1500, 2017.

[109] Mostafa Dehghani, Aliaksei Severyn, Sascha Rothe, and Jaap Kamps. Learning to learn from weak supervision by full supervision. *arXiv preprint arXiv:1711.11383*, 2017.

[110] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. Neural ranking models with weak supervision. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 65–74, New York, NY, USA, 2017. ACM.

[111] Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National Science Review*, 5(1):44–53, 2017.

[112] Geoffrey E Hinton, Terrence Joseph Sejnowski, and Tomaso A Poggio. *Unsupervised learning: foundations of neural computation.* MIT press, 1999.

[113] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[114] Anthony TC Goh. Back-propagation neural networks for modeling complex systems. *Artificial Intelligence in Engineering*, 9(3):143–151, 1995.

[115] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[116] John E Moody. The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In *Advances in neural information processing systems*, pages 847–854, 1992.

[117] Russell Reed and Robert J MarksII. *Neural smithing: supervised learning in feedforward artificial neural networks.* Mit Press, 1999.

[118] Yaochu Jin, Tatsuya Okabe, and Bernhard Sendhoff. Neural network regularization and ensembling using multi-objective evolutionary algorithms. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, volume 1, pages 1–8. IEEE, 2004.

[119] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

[120] Garvesh Raskutti, Martin J Wainwright, and Bin Yu. Early stopping and non-parametric regression: an optimal data-dependent stopping rule. *The Journal of Machine Learning Research*, 15(1):335–366, 2014.

[121] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya , and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[122] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya , and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[123] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[124] Hanna M. Wallach. Topic modeling: Beyond bag-of-words. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 977–984, New York, NY, USA, 2006. ACM.

[125] Anne Kao and Steve R Poteet. *Natural language processing and text mining.* Springer Science & Business Media, 2007.

[126] W Bruce Croft, Donald Metzler, and Trevor Strohman. *Search engines: Information retrieval in practice*, volume 520. Addison-Wesley Reading, 2010.

[127] Robert Krovetz and W Bruce Croft. Lexical ambiguity and information retrieval. *ACM Transactions on Information Systems (TOIS)*, 10(2):115–141, 1992.

[128] Burghard B Rieger. *On distributed representation in word semantics*. International Computer Science Institute Berkeley, CA, 1991.

[129] Abraham Bookstein and Don R Swanson. Probabilistic models for automatic indexing. *Journal of the American Society for Information science*, 25(5):312–316, 1974.

[130] Stephen Paul Harter. *A probabilistic approach to automatic keyword indexing*. PhD thesis, University of Chicago, 1974.

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

[132] Thomas K Landauer and Susan Dumais. Latent semantic analysis. *Scholarpedia*, 3(11):4356, 2008.

[133] Gia-Hung Nguyen, Lynda Tamine, Laure Soulier, and Nathalie Souf. Learning concept-driven document embeddings for medical information search. In *Conference on Artificial Intelligence in Medicine in Europe*, pages 160–170. Springer, 2017.

[134] Yuan Ni, Qiong Kai Xu, Feng Cao, Yosi Mass, Dafna Sheinwald, Hui Jia Zhu, and Shao Sheng Cao. Semantic documents relatedness using concept graph representation. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 635–644. ACM, 2016.

[135] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

[136] Gia-Hung Nguyen. *Modèles neuronaux pour la recherche d'information: approches dirigées par les ressources sémantiques*. PhD thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2018.

[137] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, 2017.

[138] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Charagram: Embedding words and sentences via character n-grams. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1504–1515, 2016.

[139] George Elmer Forsythe and Peter Henrici. The cyclic jacobi method for computing the principal values of a complex matrix. *Transactions of the American Mathematical Society*, 94(1):1–23, 1960.

[140] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. In *Linear Algebra*, pages 134–151. Springer, 1971.

[141] David W Miller. Computer solution of linear algebraic systems, 1968.

[142] Thomas K Landauer, Danielle S McNamara, Simon Dennis, and Walter Kintsch. *Handbook of latent semantic analysis*. Psychology Press, 2013.

[143] Thomas Hofmann. Probabilistic latent semantic indexing. In *ACM SIGIR Forum*, volume 51, pages 211–218. ACM, 2017.

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

[145] Douglas LT Rohde, Laura M Gonnerman, and David C Plaut. An improved model of semantic similarity based on lexical co-occurrence. *Communications of the ACM*, 8(627-633):116, 2006.

[146] Rémi Lebret and Ronan Collobert. Word emdeddings through hellinger pca. *arXiv preprint arXiv:1312.5542*, 2013.

[147] Andy Field. *Discovering statistics using SPSS*. Sage publications, 2009.

[148] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537, 2011.

[149] Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

[150] Ivan Vulić and Marie-Francine Moens. Monolingual and cross-lingual information retrieval models based on (bilingual) word embeddings. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 363–372. ACM, 2015.

[151] Jocelyn Coulmance, Jean-Marc Marty, Guillaume Wenzek, and Amine Benhalloum. Trans-gram, fast cross-lingual word-embeddings. *arXiv preprint arXiv:1601.02502*, 2016.

[152] Shyam Upadhyay, Manaal Faruqui, Chris Dyer, and Dan Roth. Cross-lingual models of word embeddings: An empirical comparison. *arXiv preprint arXiv:1604.00425*, 2016.

[153] Eric Nalisnick, Bhaskar Mitra, Nick Craswell, and Rich Caruana. Improving document ranking with dual word embeddings. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 83–84, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee.

[154] Hamed Zamani and W Bruce Croft. Relevance-based word embedding. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 505–514. ACM, 2017.

[155] Guangyou Zhou, Tingting He, Jun Zhao, and Po Hu. Learning continuous word embedding with metadata for question retrieval in community question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 250–259, 2015.

[156] Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering. In *Advances In Neural Information Processing Systems*, pages 289–297, 2016.

[157] Liu Yang, Qingyao Ai, Jiafeng Guo, and W Bruce Croft. anmm: Ranking short answer texts with attention-based neural matching model. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 287–296. ACM, 2016.

[158] Guoqing Zheng and Jamie Callan. Learning to reweight terms with distributed representations. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 575–584, New York, NY, USA, 2015. ACM.

[159] Debasis Ganguly, Dwaipayan Roy, Mandar Mitra, and Gareth J.F. Jones. Word embedding based generalized language model for information retrieval. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 795–798, New York, NY, USA, 2015. ACM.

[160] Fernando Diaz, Bhaskar Mitra, and Nick Craswell. Query expansion with locally-trained word embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 367–377, 2016.

[161] T Kenter, A Borisov, M de Rijke, K Erk, and NA Smith. Siamese cbow: Optimizing word embeddings for sentence representations. 2016.

[162] Felix Hill, Kyunghyun Cho, and Anna Korhonen. Learning distributed representations of sentences from unlabelled data. In *Proceedings of NAACL-HLT*, pages 1367–1377, 2016.

[163] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. 2016.

[164] Cedric De Boom, Steven Van Canneyt, Thomas Demeester, and Bart Dhoedt. Representation learning for very short texts using weighted word embedding aggregation. *Pattern Recognition Letters*, 80:150–156, 2016.

[165] Hamed Zamani and W. Bruce Croft. Estimating embedding vectors for queries. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, ICTIR '16, pages 123–132, New York, NY, USA, 2016. ACM.

[166] Tiancheng Zhao, Kyusong Lee, and Maxine Eskenazi. Unsupervised discrete sentence representation learning for interpretable neural dialog generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1098–1107, 2018.

[167] Lajanugen Logeswaran and Honglak Lee. An efficient framework for learning sentence representations. 2018.

[168] Wenpeng Yin and Hinrich Schütze. Convolutional neural network for paraphrase identification. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 901–911, 2015.

[169] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[170] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[171] Francis Jeffry Pelletier. The principle of semantic compositionality. *Topoi*, 13(1):11–24, 1994.

[172] Qingyao Ai, Liu Yang, Jiafeng Guo, and W. Bruce Croft. Improving language estimation with the paragraph vector model for ad-hoc retrieval. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 869–872, New York, NY, USA, 2016. ACM.

[173] Lingfei Wu, Ian En-Hsu Yen, Kun Xu, Fangli Xu, Avinash Balakrishnan, Pin-Yu Chen, Pradeep Ravikumar, and Michael J Witbrock. Word mover's embedding: From word2vec to document embedding. pages 4524–4534, 2018.

[174] Gao Huang, Chuan Guo, Matt J Kusner, Yu Sun, Fei Sha, and Kilian Q Weinberger. Supervised word mover's distance. In *Advances in Neural Information Processing Systems*, pages 4862–4870, 2016.

[175] Eunjeong L Park, Sungzoon Cho, and Pilsung Kang. Supervised paragraph vector: distributed representations of words, documents and class labels. *IEEE Access*, 2019.

[176] Tom Kenter and Maarten de Rijke. Short text similarity with word embeddings. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 1411–1420, New York, NY, USA, 2015. ACM.

[177] Qingyao Ai, Liu Yang, Jiafeng Guo, and W. Bruce Croft. Analysis of the paragraph vector model for information retrieval. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, ICTIR '16, pages 133–142, New York, NY, USA, 2016. ACM.

[178] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In *25th ACM International Conf. CIKM'16*, 2016.

[179] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 1291–1299, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.

[180] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 55–64, New York, NY, USA, 2017. ACM.

[181] Guido Zuccon, Bevan Koopman, Peter Bruza, and Leif Azzopardi. Integrating and evaluating neural word embeddings in information retrieval. In *Proceedings of the 20th Australasian document computing symposium*, page 12. ACM, 2015.

[182] Adam Berger and John Lafferty. Information retrieval as statistical translation. *SIGIR Forum*, 51(2):219–226, August 2017.

[183] Guoqing Zheng and Jamie Callan. Learning to reweight terms with distributed representations. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 575–584, New York, NY, USA, 2015. ACM.

[184] Bhaskar Mitra, Eric Nalisnick, Nick Craswell, and Rich Caruana. A dual embedding space model for document ranking. *arXiv preprint arXiv:1602.01137*, 2016.

[185] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In *23rd International Conference on World Wide Web'14*, 2014.

[186] Matthew Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. Dissecting contextual word embeddings: Architecture and representation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, 2018.

[187] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, 2018.

[188] Kamal Al-Sabahi and Zhang Zuping. Document summarization using sentence-level semantic based on word embeddings. *International Journal of Software Engineering and Knowledge Engineering*, 29(02):177–196, 2019.

[189] Melvin Earl Maron and John L Kuhns. On relevance, probabilistic indexing and information retrieval. *Journal of the ACM (JACM)*, 7(3):216–244, 1960.

[190] Dwaipayan Roy, Debjyoti Paul, Mandar Mitra, and Utpal Garain. Using word embeddings for automatic query expansion. *arXiv preprint arXiv:1606.07608*, 2016.

[191] Saar Kuzi, Anna Shtok, and Oren Kurland. Query expansion using word embeddings. In *Proceedings of the 25th ACM international on conference on information and knowledge management*, pages 1929–1932. ACM, 2016.

[192] Navid Rekabsaz, Mihai Lupu, Allan Hanbury, and Hamed Zamani. Word embedding causes topic shifting; exploit global context! In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 1105–1108, New York, NY, USA, 2017. ACM.

[193] Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012.

[194] Navid Rekabsaz, Bhaskar Mitra, Mihai Lupu, and Allan Hanbury. Toward incorporation of relevant documents in word2vec. *ArXiv*, abs/1707.06598, 2017.

[195] Kaveh Taghipour and Hwee Tou Ng. Semi-supervised word sense disambiguation using word embeddings in general and specific domains. In *Proceedings of the 2015 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 314–323, 2015.

[196] Oded Avraham and Yoav Goldberg. The interplay of semantics and morphology in word embeddings. *EACL 2017*, page 422, 2017.

[197] Ignacio Iacobacci, Mohammad Taher Pilehvar, and Roberto Navigli. Sensembed: Learning sense embeddings for word and relational similarity. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 95–105, 2015.

[198] Yadollah Yaghoobzadeh and Hinrich Schütze. Intrinsic subspace evaluation of word embedding representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 236–246, 2016.

[199] Nikola Mrkšic, Diarmuid OSéaghdha, Blaise Thomson, Milica Gašic, Lina Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. Counter-fitting word vectors to linguistic constraints. In *Proceedings of NAACL-HLT*, pages 142–148, 2016.

[200] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. Contextualized word representations for document re-ranking. *arXiv preprint arXiv:1904.07094*, 2019.

[201] Zhuyun Dai and Jamie Callan. Deeper text understanding for ir with contextual neural language modeling. *arXiv preprint arXiv:1905.09217*, 2019.

[202] Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R Bowman, Dipanjan Das, et al. What do you learn from context? probing for sentence structure in contextualized word representations. *arXiv preprint arXiv:1905.06316*, 2019.

[203] Yifan Qiao, Chenyan Xiong, Zhenghao Liu, and Zhiyuan Liu. Understanding the behaviors of bert in ranking. *arXiv preprint arXiv:1904.07531*, 2019.

[204] Dwaipayan Roy, Debasis Ganguly, Sumit Bhatia, Srikanta Bedathur, and Mandar Mitra. Using word embeddings for information retrieval: How collection and term normalization choices affect performance. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1835–1838. ACM, 2018.

[205] Hang Li. Learning to rank for information retrieval and natural language processing. *Synthesis Lectures on Human Language Technologies*, 4(1):1–113, 2011.

[206] Sally Jo Cunningham, James Littin, and Ian H Witten. Applications of machine learning in information retrieval. 1997.

[207] Tie-Yan Liu. *Learning to rank for information retrieval*. Springer Science & Business Media, 2011.

[208] Kevin Duh and Katrin Kirchhoff. Learning to rank with partially-labeled data. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 251–258. ACM, 2008.

[209] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136. ACM, 2007.

[210] Christopher Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine learning (ICML-05)*, pages 89–96, 2005.

[211] Wei Chen, Tie-Yan Liu, Yanyan Lan, Zhi-Ming Ma, and Hang Li. Ranking measures and loss functions in learning to rank. In *Advances in Neural Information Processing Systems*, pages 315–323, 2009.

[212] Martin Szummer and Emine Yilmaz. Semi-supervised learning to rank with preference regularization. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 269–278. ACM, 2011.

[213] Tao Qin and Tie-Yan Liu. Introducing letor 4.0 datasets. *arXiv preprint arXiv:1306.2597*, 2013.

[214] Wei Chu and Zoubin Ghahramani. Gaussian processes for ordinal regression. *Journal of machine learning research*, 6(Jul):1019–1041, 2005.

[215] Wei Chu and S Sathiya Keerthi. New approaches to support vector ordinal regression. In *Proceedings of the 22nd international conference on Machine learning*, pages 145–152. ACM, 2005.

[216] William S Cooper, Fredric C Gey, and Daniel P Dabney. Probabilistic retrieval based on staged logistic regression. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 198–210. ACM, 1992.

[217] Brian Bartell, Garrison W Cottrell, and Richard Belew. Learning to retrieve information. In *Proceedings of the Swedish Conference on Connectionism*, page 27. Citeseer, 1995.

[218] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. Adapting ranking svm to document retrieval. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 186–193. ACM, 2006.

[219] Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, Xu-Dong Zhang, and Hang Li. Learning to search web pages with query-level loss functions. *Technical Report*, 156, 2006.

[220] Tao Qin, Xu-Dong Zhang, Ming-Feng Tsai, De-Sheng Wang, Tie-Yan Liu, and Hang Li. Query-level loss functions for information retrieval. *Information Processing & Management*, 44(2):838–855, 2008.

[221] Xiubo Geng, Tie-Yan Liu, Tao Qin, and Hang Li. Feature selection for ranking. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 407–414. ACM, 2007.

[222] Hua-Liang Wei and Stephen A Billings. Feature subset selection and ranking for data dimensionality reduction. *IEEE transactions on pattern analysis and machine intelligence*, 29(1):162–166, 2006.

[223] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.

[224] Bhaskar Mitra, Nick Craswell, et al. An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval*, 13(1):1–126, 2018.

[225] Nick Craswell, W Bruce Croft, Jiafeng Guo, Bhaskar Mitra, and Maarten de Rijke. Report on the sigir 2016 workshop on neural information retrieval (neu-ir). In *ACM Sigir forum*, volume 50, pages 96–103. ACM, 2017.

[226] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1291–1299. International World Wide Web Conferences Steering Committee, 2017.

[227] Hang Li and Zhengdong Lu. Deep learning for information retrieval. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 1203–1206, New York, NY, USA, 2016. ACM.

[228] Kezban Dilek Onal, Ye Zhang, Ismail Sengor Altingovde, Md Mustafizur Rahman, Pinar Karagoz, Alex Braylan, Brandon Dang, Heng-Lu Chang, Henna Kim, Quinten McNamara, Aaron Angert, Edward Banner, Vivek Khetan, Tyler McDonnell, An Thanh Nguyen, Dan Xu, Byron C. Wallace, Maarten de Rijke, and Matthew Lease. Neural information retrieval: at the end of the early years. *Information Retrieval Journal*, 21(2):111–182, Jun 2018.

[229] Rudolf Schneider, Sebastian Arnold, Tom Oberhauser, Tobias Klatt, Thomas Steffek, and Alexander Löser. Smart-md: Neural paragraph retrieval of medical topics. In *Companion Proceedings of the The Web Conference 2018*, pages 203–206. International World Wide Web Conferences Steering Committee, 2018.

[230] Lei Yu, Karl Moritz Hermann, Phil Blunsom, and Stephen Pulman. Deep learning for answer sentence selection. *arXiv preprint arXiv:1412.1632*, 2014.

[231] Jinfeng Rao, Hua He, and Jimmy Lin. Noise-contrastive estimation for answer selection with deep neural networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 1913–1916, New York, NY, USA, 2016. ACM.

[232] Bingning Wang, Kang Liu, and Jun Zhao. Inner attention based recurrent neural networks for answer selection. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1288–1297, 2016.

[233] Xipeng Qiu and Xuanjing Huang. Convolutional neural tensor network architecture for community-based question answering. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[234] Xiaoqiang Zhou, Baotian Hu, Qingcai Chen, Buzhou Tang, and Xiaolong Wang. Answer sequence learning with neural networks for answer selection in community question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 713–718, 2015.

[235] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[236] Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *Transactions of the Association for Computational Linguistics*, 4:259–272, 2016.

[237] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*, pages 101–110. ACM, 2014.

[238] Aliaksei Severyn and Alessandro Moschitti. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 373–382. ACM, 2015.

[239] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 24(4):694–707, 2016.

[240] Peixin Chen, Wu Guo, Zhi Chen, Jian Sun, and Lanhua You. Gated convolutional neural network for sentence matching. *Proc. Interspeech 2018*, pages 2853–2857, 2018.

[241] Seonhoon Kim, Inho Kang, and Nojun Kwak. Semantic sentence matching with densely-connected recurrent and co-attentive information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6586–6593, 2019.

[242] Zhengdong Lu and Hang Li. A deep architecture for matching short texts. In *Advances in neural information processing systems*, pages 1367–1375, 2013.

[243] Sanjay Kamath, Brigitte Grau, and Yue Ma. Predicting and Integrating Expected Answer Types into a Simple Recurrent Neural Network Model for Answer Sentence Selection. In *20th International Conference on Computational Linguistics and Intelligent Text Processing*, La Rochelle, France, April 2019.

[244] Xiaoqiang Zhou, Baotian Hu, Qingcai Chen, and Xiaolong Wang. Recurrent convolutional neural network for answer selection in community question answering. *Neurocomputing*, 274:8–18, 2018.

[245] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR conference on research and development in information retrieval*, pages 55–64. ACM, 2017.

[246] Shengxian Wan, Yanyan Lan, Jun Xu, Jiafeng Guo, Liang Pang, and Xueqi Cheng. Match-srnn: Modeling the recursive matching structure with spatial rnn. *arXiv preprint arXiv:1604.04378*, 2016.

[247] Kai Hui, Andrew Yates, Klaus Berberich, and Gerard de Melo. Position-aware representations for relevance matching in neural information retrieval. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, pages 799–800, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.

[248] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 655–665, 2014.

[249] Daniel Cohen, Qingyao Ai, and W Bruce Croft. Adaptability of neural networks on varying granularity ir tasks. *arXiv preprint arXiv:1606.07565*, 2016.

[250] Liu Tieyan, Qin Tao, Xu Jun, et al. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of the Workshop on Learning to Rank for Information Retrieval*, pages 137–145, 2007.

[251] Hamed Zamani, Bhaskar Mitra, Xia Song, Nick Craswell, and Saurabh Tiwary. Neural ranking models with multiple document fields. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 700–708. ACM, 2018.

[252] Thiziri Belkacem, Jose G Moreno, Taoufiq Dkaki, and Mohand Boughanem. Asymmetry sensitive architecture for neural text matching. In *European Conference on Information Retrieval*, pages 62–69. Springer, 2019.

[253] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.

[254] Trevor Strohman, Donald Metzler, Howard Turtle, and W Bruce Croft. Indri: A language model-based search engine for complex queries. In *Proceedings of the International Conference on Intelligent Analysis*, volume 2, pages 2–6. Citeseer, 2005.

[255] Christophe Van Gysel, Evangelos Kanoulas, and Maarten de Rijke. Pyndri: a python interface to the indri search engine. In *European Conference on Information Retrieval*, pages 744–748. Springer, 2017.

[256] C Buckley. trec_eval program, march 1999. *Available from ftp. cs. cornell, edu/pub/smart*, 1975.

[257] Hamed Zamani and W Bruce Croft. Embedding-based query language models. In *Proceedings of the 2016 ACM international conference on the theory of information retrieval*, pages 147–156. ACM, 2016.

[258] Yuanhua Lv and ChengXiang Zhai. Lower-bounding term frequency normalization. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 7–16. ACM, 2011.

[259] Thiziri Belkacem, Taoufiq Dkaki, José G. Moreno, and Mohand Boughanem. Impact de la présence/absence des termes de la requête dans le document sur le processus d'appariement document-requête en utilisant word2vec. In *COnférence en Recherche d'Informations et Applications - CORIA 2018, 15th French Information Retrieval Conference, Rennes, France, May 16-18, 2018. Proceedings.*, 2018.

[260] Dwaipayan Roy, Debasis Ganguly, Mandar Mitra, and Gareth JF Jones. Representing documents and queries as sets of word embedded vectors for information retrieval. 2016.

[261] Navid Rekabsaz, Mihai Lupu, and Allan Hanbury. Exploration of a threshold for similarity based on uncertainty in word embedding. In Joemon M Jose, Claudia Hauff, Ismail Sengor Altıngovde, Dawei Song, Dyaa Albakour, Stuart Watt, and John Tait, editors, *Advances in Information Retrieval*, pages 396–409, Cham, 2017. Springer International Publishing.

[262] Geoffrey Zweig, John C. Platt, Christopher Meek, Christopher J. C. Burges, Ainur Yessenalina, and Qiang Liu. Computational approaches to sentence completion. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 601–610, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[263] Yoon Kim. Convolutional neural networks for sentence classification. pages 1746–1751, 2014.

[264] K Abishek, Basuthkar Rajaram Hariharan, and C Valliyammai. An enhanced deep learning model for duplicate question pairs recognition. In *Soft Computing in Data Analytics*, pages 769–777. Springer, 2019.

[265] Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer, 2005.

[266] Till Haug, Octavian-Eugen Ganea, and Paulina Grnarova. Neural multi-step reasoning for question answering on semi-structured tables. In *European Conference on Information Retrieval*, pages 611–617. Springer, 2018.

[267] Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. Text classification improved by integrating bidirectional lstm with two-dimensional max pooling. *arXiv preprint arXiv:1611.06639*, 2016.

[268] Kateryna Tymoshenko and Alessandro Moschitti. Cross-pair text representations for answer sentence selection. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2162–2173, 2018.

[269] James Allan, Jay Aslam, Nicholas Belkin, Chris Buckley, Jamie Callan, Bruce Croft, Sue Dumais, Norbert Fuhr, Donna Harman, David J Harper, et al. Challenges in information retrieval and language modeling: report of a workshop held at the center for intelligent information retrieval, university of massachusetts amherst, september 2002. In *ACM SIGIR Forum*, volume 37, pages 31–47. ACM, 2003.

[270] Andreas Zell. *Simulation neuronaler netze*, volume 1. Addison-Wesley Bonn, 1994.

[271] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[272] Thiziri Belkacem, Taoufiq Dkaki, Jose G Moreno, and Mohand Boughanem. amv-lstm: an attention-based model with multiple positional text matching. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 788–795. ACM, 2019.

[273] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, 2016.

[274] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. Ms marco: A human-generated machine reading comprehension dataset. 2016.