# Université de Montréal


# Tailored Deep Learning Techniques for Information Retrieval


par

**Yifan Nie**

Département d'Informatique et Recherches Opérationnelles
Faculté des arts et des sciences


Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique

December 31, 2021

# Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

**Tailored Deep Learning Techniques for Information Retrieval**

présentée par

## Yifan Nie

a été évaluée par un jury composé des personnes suivantes :

*Philippe Langlais*
(président-rapporteur)

*Jian-Yun Nie*
(directeur de recherche)

*Bang Liu*
(membre du jury)

*Jianfeng Gao*
(examinateur externe)

*Lubna Daraz*
(représentant du doyen de la FESP)

# Résumé

La recherche d'information vise à trouver des documents pertinents par rapport à une requête. Auparavant, de nombreux modèles traditionnels de la Recherche d'Informations ont été proposés. Ils essaient soit d'encoder la requête et les documents en vecteurs dans l'espace des termes et d'estimer la pertinence en calculant la similarité des deux vecteurs, soit d'estimer la pertinence par des modèles probabilistes. Cependant, pour les modèles d'espace vectoriel, l'encodage des requêtes et des documents dans l'espace des termes a ses limites: par exemple, il est difficile d'identifier les termes du document qui ont des sens similaires au termes exactes de la requête. Il est également difficile de représenter le contenu du texte à différents niveaux d'abstraction pouvant correspondre aux besoins différents d'information exprimés dans des requêtes. Avec le développement rapide des techniques d'apprentissage profond, il est possible d'apprendre des représentations utiles à travers une série de couches neurones, ce qui ouvre la voie à de meilleures représentations dans un espace dense latent plutôt que dans l'espace des termes, ce qui peut aider à identifier les termes non exactes mais qui portent les sens similaires. Il nous permet également de créer de différentes couches de représentation pour la requête et le document, permettant ainsi des correspondances entre la requête et les documents à différents niveaux d'abstractions, ce qui peut mieux répondre aux besoins d'informations pour différents types de requêtes. Enfin, les techniques d'apprentissage profond permettent également d'apprendre une meilleure fonction d'appariement.

Dans cette thèse, nous explorons différentes techniques d'apprentissage profond pour traiter ces problèmes. Nous étudions d'abord la construction de plusieurs couches de représentation avec différents niveaux d'abstraction entre la requête et le document, pour des modèles basés sur la représentation et l'interaction. Nous proposons ensuite un modèle permettant de faire les matchings croisés des representations entre la requête et le document sur différentes couches pour mieux répondre au besoin de correspondance terme-phrase. Enfin, nous explorons l'apprentissage intégré d'une fonction de rang et les représentations de la requête et du document. Des expériences sur des jeux de données publics ont montré que nos méthods proposées dans cette thèse sont plus performantes que les méthodes existantes.

**Mots-clés: Recherche d'Information; Apprentissage Profond; Réseaux Neurones; Tri**

# Abstract

Information Retrieval aims to find relevant documents to a query. Previously many traditional information retrieval models have been proposed. They either try to encode query and documents into vectors in term space and estimate the relevance by computing the similarity of the two vectors or estimate the relevance by probabilistic models. However for vector space models, encoding query and documents into term space has its limitations: for example, it's difficult to catch terms of similar meanings to the exact query term in the document. It is also difficult to represent the text in a hierarchy of abstractions to better match the information need expressed in the query. With the fast development of deep learning techniques, it is possible to learn useful representations through a series of neural layers, which paves the way to learn better representations in latent dense space rather the term space, which may help to match the non exact matched but similar terms. It also allows us to create different layers of representation for query and document thereby enabling matchings between query and documents at different levels of abstractions, which may better serve the information needs for different queries. Finally, deep learning techniques also allows to learn better ranking function.

In this thesis, we explore several deep learning techniques to deal with the above problems. First, we study the effectiveness of building multiple abstraction layers between query and document, for representation- and interaction-based models. Then we propose a model allowing for cross-matching of query and document representations at different layers to better serve the need of term-phrase matching. Finally we propose an integrated learning framework of ranking function and neural features from query and document. Experiments on public datasets demonstrate that the methods we propose in this thesis are more effective than the existing ones.

**Key Words: Information Retrieval; Deep Learning; Neural Networks; Ranking**

# Contents

# List of tables

# List of figures

# Liste des sigles et des abréviations

| | |
|---|---|
| IR | Information Retrieval |
| IDF | Inverse Document Frequency |
| L2R | Learning-to-rank |
| LM | Language Model |
| MAP | Mean Average Precision |
| nDCG | Normalized Discounted Cumulative Gain |
| VSM | Vector Space Model |

# Remerciements

I would like to first thank my supervisor, Prof. Jian-Yun Nie, for his generous help, support and constructive advice during my study. I learned a lot from Prof. Nie on how to do good research and how to present the research results in a focused and concise manner. I also want to express my gratitude to my lab mates Peng Lu, David Alfonso Hermelo, Ting Bai with whom I had wonderful discussions on visions and technical details of my research. I also want to thank the research associates in our lab, Fabrizio Gotti, Pan Du and other professors who helped me during my study. Finally I want to thank my parents who supported me wholeheartedly throughout my study.

# Chapter 1

# Introduction

## 1.1. Research Context

### 1.1.1. Background

Information Retrievel (IR) plays an important role in our life. Given a query, the goal of information retrieval is to retrieve a ranked list of relevant documents [Drott, 1998, Hiemstra, 2001]. When a query is issued, the IR system will analyze the query and try to understand the information need expressed in this query as well as the documents in a document collection [den Branden, 2007]. In order to achieve this, the system will often convert the query and documents into certain representations and then feed the representations of query and documents into a ranking function to calculate their relevance score [Croft, 1981, Belkin et al., 1995]. Finally the system will rank the retrieved documents according to their relevance scores in descending order and present those results to the user.

To accomplish this task, many traditional approaches have been proposed, which could be roughly divided into Vector Space Model (VSM) [Lee et al., 1997, Castells et al., 2007], Probabilistic Models [Fuhr, 1992, Jones et al., 2000a,b] and Language Models [Zhai and Lafferty, 2001, Tao et al., 2006]. Vector Space Models encode a query and a document into vectors [Soucy and Mineau, 2005]. Most of them use a bag-of-words representation [Wallach, 2006, Sethy and Ramabhadran, 2008] to encode query and document into weighted vectors in term space [Wu et al., 2008]. Finally a similarity function such as dot product or cosine similarity [Sidorov et al., 2014] is applied between the query and document representation vectors to estimate the relevance of query and document. On the other hand, Probabilistic Model relies on an estimation of the probability of relevance for a given query and document [Porter, 1982, Crestani et al., 1998]. Finally, Language Model method tries to estimate a language model for each document and rank documents by the likelihood of the query according to the language model [Zhai and Lafferty, 2001].

Although those traditional models have demonstrated their effectiveness, they still face challenges because the features employed are often superficial in the vocabulary space, and could only catch exact term matches between query and document. Those approaches fail to match important information expressed by semantically related terms in the document. Very often, to satisfy the user's information need expressed in a query, it requires more than a simple term matching and the matching is highly semantic and conceptual, which the traditional models are unable to achieve. For example, for the query "lymphoma in dogs", the document containing "canine blood cancer" is also relevant, since lymphoma is a type of cancer, and dogs belong to the canine category. There's no exact-matched terms between this particular query and document. More importantly, lymphoma is not a synonym of cancer but is a special type of cancer and therefore implies the concept of "cancer". The same goes for dog and canine. Therefore a retrieval model needs to generalize the semantics of query terms "lymphoma" and "dog" into the concept space in order to successfully match them with relevant documents. This is difficult for a traditional model to achieve without using external resources such as concept nets [Egozi et al., 2011] where related concepts such as dog and canine are connected through a semantic relation. Intuitively, much of such general knowledge should have been incorporated in the representations, so that a more specific term would have some commonality in its representation with a more general term. This is difficult to achieve with the bag-of-words representations. From this example, we can observe that there is also the need to learn a hierarchy of representations at different levels of abstraction for queries and documents in IR tasks and generalize query and document to the appropriate level when needed.

Recently, with the rapid development of deep learning techniques [Bengio, 2009, LeCun et al., 2015], it is possible to learn semantic and abstract representations from raw texts to some extent [Boom et al., 2016, Zhang et al., 2017a,b]. This opens a new perspective for IR to perform semantic-based retrieval. Typically, those models employ neural networks to learn a series of layers for query and document, and utilize the last layer's vector to calculate a relevance score. Depending on how the neural layered are learned, those neural models could be roughly categorized into representation- and interaction-based model.

Representation-based models learn semantic representations for query and document separately through a series of neural representation learning layers and employs the last layer of query and document representations to output a relevance score. For example, the DSSM [Huang et al., 2013] employs feed-forward layers to encode query and document and applies a cosine similarity function on the last layer of query and document representations to estimate the relevance score. CDSSM [Shen et al., 2014] employs 1D convolution to learn query and document representations, and applies a cosine similarity on the last layer of query and document representation to obtain a global matching score.

On the other hand, interaction-based models build local interaction pattern between query and document at the bottom and then employ multiple neural layers to analyze it. Finally the last layer of the interaction pattern will be used to output a score. For instance, the MatchPyramid model [Pang et al., 2016a] builds local term interactions between query and document by cosine similarity at the bottom and employs 2D convolutions to analyze the interaction pattern. Finally it outputs a relevance score based on the information obtained on the last max-pooled layer. Another way to analyze the matching pattern is to employ 2D-RNNs. MatchSRNN [Wan et al., 2016] first built term-to-term local interactions in a similar way as MatchPyramid. Afterwards, instead of employing 2D-CNN to analyze the interaction pattern, it utilizes a 2D-RNN to read the interactions in two directions: from left to right and from top to bottom. Finally a series of dense layers are employed to output a global matching score. The DRMM model [Guo et al., 2016] employs histograms to represent the interaction pattern between query and document. It first calculates term-wise cosine similarity between each query term and all document terms. Afterwards it converts these interaction values into histograms according to the intervals that they fall into. Finally, dense layers are applied to analyze the interaction patterns and a gating mechanism is employed to combine the signals for all query terms.

It is worth noting that all these previous deep models have one common feature, that is: they only employ the information learned at the last layer to represent the query and document or the interactions between them. In Chapter 2, we will provide a more detailed presentations of these approaches.

In addition to those models, another framework called Learning-to-Rank (L2R) is widely used in the current IR research and practice [Liu, 2011, Cao et al., 2007]. In L2R, relevance estimation is regarded as a machine learning task, where the model will typically employ a wide range of features [Chapelle and Chang, 2011, Li, 2011] ranging from different relevance scores determined by traditional models [Robertson et al., 2004, Clinchant and Gaussier, 2010], to text statistics such as the length of document title, document body, anchor text [Liu, 2009] and webpage specific scores like PageRank [Page et al., 1999, Singh and Kumar, 2009] and HITS scores [Henzinger, 2000, Peserico and Pretto, 2009]. The goal is to learn implicitly a combination function which combines all those features and estimate the relevance of query and document. However, traditional L2R framework takes fixed feature as input. Within the deep learning context, it is useful to incorporate representations and interactions learned from neural models as features and use L2R ranking function to fine-tune the neural layers.

### 1.1.2. Limitations of the Existing Models

To summarize, in order to better apply deep learning techniques to IR models, we need to improve the following aspects in the current approaches:

**1.** Firstly, when multiple layers are involved in a neural network, one can generally assume that a lower layer captures specific and local information, while a higher layer captures more general and global information from a text. In previous work, the query and document representations used to produce a matching score are usually the last layer of the deep neural model, which contains highly abstract information such as concepts and semantics of the raw text or the abstract interaction pattern. However in information retrieval, the information need formulated in user's query is not always at a high abstract or conceptual level. It varies from lexical matching to conceptual matching as we explain below.

Some queries may ask for an exact word match to retrieve a document containing specific terms. We call those queries lexical queries. In this case, representation learned in the lower part of the model would be more suitable. For example, in the query "Ron Howard", what the user wants to find is the document concerning this specific person. If an IR model, as previously mentioned, always utilizes the last representation layer or interaction pattern to perform a match, it may over generalize the lexical terms and find documents about other people.

Other queries ask for a match at more abstract/conceptual level, where the use of high level representations is required. For example, if the user's query is "terminate employment letter", the user's actual information need will be searching for a template of a dismissal letter to fire someone. In this case, the documents containing terms like "fire", "dismiss" are all relevant to this query, because the actual information need expressed in the query is about the concept of "employment/dismissal". Hence the utilization of higher layers are preferred, since they contain generalized semantics which could help to match those terms in concept space.

Finally there could well be queries lying in between. Therefore we argue that the use of features at all levels could be helpful to further improve the performance of deep learning IR models. Moreover the model should have the capability to dynamically adjust itself to adapt to all types of query during the run.

**2.** Secondly, apart from employing multiple layers of the representations or interactions of query and document from the same level, it is also useful to allow matchings to happen across different levels of representations. We observe that in previous interaction-based models, the matching often happens on the term level. For example, in MatchPyramid model [Pang et al., 2016a], the interaction matrix is generated by matching query and document term embeddings. In DRMM model [Guo et al., 2016], the histogram which represents the interaction signals between query and documents is also produced by matching query term embeddings with document term embeddings.

In reality, a query term can also match a phrase or even a longer segment of text in documents and vice versa. For instance, for the query "what's the standard barrel length for an AR", it is helpful to match the query phrase "standard barrel length" with the terms

"AK47", "Carbine" in relevant documents, since they are all related to guns. Similarly, for the query "what's the nickname for Nevada", it is also helpful to match the query term "Nevada" to the document phrase "Silver State", because they both refer to the State of Nevada. Finally, for the query "stainless steel", it is also beneficial to match it with the document containing the phrase "corrosion resistance" since "corrosion resistance" is one of the properties of stainless steel.

In a neural model, the representation learning layers naturally aggregate low-level term elements into higher-level hidden representations, which enables us to map term representations into phrase representations and match them to provide additional matching signals to the model. This paves the way for performing term-phrase matching to serve the aforementioned need. The same cross-level matching may happen between any pair of representation layers. Therefore it is worthwhile to investigate the possibility of matching not only representations of the same level, but also those across different levels.

**3.** Thirdly, we observe a large gap between traditional learning-based IR model such as L2R and neural IR model on several test collections. Neural models do not always perform better than the traditional L2R models. Usually, when training a neural IR model, we focus on learning appropriate representations for queries and documents [Guo et al., 2016]. However, when it comes to the ranking function, very often those models will employ a simple pair-wise hinge loss [Pang et al., 2016a, Xiong et al., 2017b] to train. Employing a simple hinge loss is straightforward and effective. However, in such a ranking task, it is desirable to have relevant documents ranked higher and non relevant documents ranked lower in the ranked list. Therefore, it is preferable to train a neural model in a L2R framework which could push relevant documents upwards and non relevant documents downwards in the ranked list.

When it comes to training a L2R model, we focus on learning a ranking function based on all the features of the query and documents. However the input features are often fixed hand-crafted features such as query length, document length, term frequencies, BM25 scores of query-document pair etc. Those simple hand-crafted features lack the expressiveness of learned neural representations.

To take advantage of both the representation learning power of deep models and the ranking performance of L2R framework, it is natural to investigate whether it is possible to combine deep learning model and L2R together to learn the representation and the ranking function at the same time. The combination of L2R with deep neural representations could be done quite easily as several L2R models are based on neural networks such as RankNet [Burges et al., 2005] and LambdaRank [Burges et al., 2006]. It is thus possible to design a combined architecture where the deep learning module is in charge of representation learning and the upper L2R part is in charge of learning a ranking function.

In summary, deep learning approach for IR presents a lot of potential advantages over traditional IR approaches, nevertheless, it still faces challenges. By thoroughly investigating the above aspects, we hope to be able to further improve the performance of deep learning models in IR.

## 1.2. Outline of the Thesis

The remainder of this thesis is organized as follows:

Chapter 2 will present the related work and the state-of-the-art approaches. We will present traditional IR models including Vector Space Models and Probabilistic Models, including the well-known and widely used BM25 model [Robertson et al., 2004] and Language Model (LM) [Song and Croft, 1999, Zhai and Lafferty, 2001]. We then present the traditional learning-to-rank framework. Finally, we present some of the state-of-the-art neural IR models, including DSSM , CDSSM, MatchPyramid and DRMM.

Chapter 3 will present the evaluation methods and the collections utilized in this study.

Chapter 4 deals with multi-level document and query representations. We will present the Multi-level Abstraction Matching Model (MACM) with Weak Supervision for Information Retrieval and an Empirical study of MACM based on representations and interactions. These investigations aim to answer the first research question raised in Section 1.1:

**Is employing multiple levels of representations or interactions between query and document useful for an neural retrieval model? If yes, how to combine the maching signals from multiple levels?**

Inspired by MatchPyramid [Pang et al., 2016a] and CDSSM [Shen et al., 2014], the MACM models employ 2D or 1D convolutions to learn interactions between query and document or representations of them. Instead of using only the last layer's interaction pattern or representation, we extract interaction or representation signals from every layer and employ a gating mechanism to dynamically combine them to output a relevance score. Experiments on the ClueWeb09 collection demonstrated the usefulness of using matching signals from multiple matching layers.

Chapter 5 investigates cross-level matching and we propose a Cross-level Matching Model for Information Retrieval (CLMM). This addresses the second research question raised in Section 1.1:

**Is it helpful to cross-match different layers of query and document representations to serve the need of term-phrase matching and improve the performance of a neural IR model?**

In this chapter, we proposed a CLMM model which employs 1D convolution and bidirectional LSTM to aggregate low level term representations into phrase representations. Afterwards cross-level matchings generate term-to-term, term-to-phrase, phrase-to-phrase

matching patterns. Those matching patterns are fed into 2D convolutions to be analyzed. Finally a gating mechanism dynamically combines the matching signals from all channels. Experiments on MSMARCO dataset demonstrated the effectiveness of employing cross-level matching signals to improve the performance for IR.

Chapter 6 addresses the problem of combining deep neural representations with L2R. We propose an Integrated Learning of Features and Ranking Function for Information Retrieval (ILM). This aims to answer the third research question in Section 1.1:

**Is it advantageous to combine both neural representation modules and L2R framework together in order to learn features and ranking function simultaneously? If yes, how to effectively combine them?**

The proposed model learns features of query and document and a ranking function end-to-end by a LambdaRank loss function. We first learn neural features from query and document texts by neural representation and interaction modules. They are then connected to a L2R layer. In this layer, an arbitrary set of additional non neural features could also be appended. Finally we employ LambdaRank loss to train both L2R layer and neural feature learning modules in an end-to-end manner. Experiments on MQ datasets revealed its potential over learning features and ranking function separately.

Chapter 7 will give a general conclusion of this thesis and point out possible future research directions.

# Chapter 2

# Related Work

In IR literature, many information retrieval systems have been built and lots of approaches have been proposed. In this chapter, we describe the most representative and related traditional IR models and deep learning IR models and discuss about the state of the art models in IR.

## 2.1. Traditional Retrieval Model

The ultimate goal of traditional models is to be able to estimate a relevance score given the query and document as closely as possible to the user's own judgment. The traditional models can be roughly divided into 2 categories, i.e. vector space model (VSM) and probabilistic model (PM) [Croft et al., 2009, Roelleke, 2013]. Vector space models (VSM) generally convert queries and documents into bag-of-words vector representations [Raghavan and Wong, 1986], i.e. the vector contains counts or other weights in the term space, in which each dimension represents a term. The relevance score of a given query and document pair is often calculated with scalar product based cosine similarity. On the other hand, probabilistic model tries to model the probability $P(R = 1|q,d)$ [Singhal, 2001, Clinchant and Gaussier, 2012], i.e. regarding the relevance R as a random variable taking values from {0,1} and modeling the probability of the relevant being 1 given query $q$ and document $d$ [Bikel and Zitouni, 2012].

Later, in order to further improve the performance of IR, learning-to-rank (L2R) approaches are proposed. L2R approaches regard IR task as a supervised learning process, and take features like basic relevance scores such as BM25 [Robertson et al., 2004], LM [Hiemstra, 2001], other text statistics such as document length, and webpage specific scores such as PageRank and HITS scores [Liu, 2009]. The following sections will briefly review those traditional models and discuss their advantages and limitations.

### 2.1.1. Vector Space Model

The history of bag-of-words representation started with the well known vector space model (VSM) [Salton, 1971] which represents queries and documents by vectors in the term space [Croft et al., 2009]. In vector space models, query and document will be encoded into bag-of-words vectors by a weighting scheme. A simple weighting scheme uses only term frequencies (TF) for both query and document vectors. However this simple weighting scheme is not optimal because it will be dominated by frequent words in a language such as "new", "file", etc. which are not very informative about the content of a document. Thus it is important to penalize those frequent words in documents [Fang et al., 2004]. The most common approach is the TF-IDF weighting scheme. In TF-IDF weighting scheme [Aizawa, 2003], each element of the document vector representation is calculated as the product of its term frequency (TF) and inverse document frequency (IDF), by Equation 2.1.1. [Ramos et al., 2003].

$$TF - IDF_{ij} = tf_{ij} \cdot log\frac{N}{n_i} \tag{2.1.1}$$

where $tf_{ij}$ represent the term frequency of term $i$ in document $j$, $N$ represent the total number of documents in the collection, and $n_i$ represents the number of documents containing the term $i$.

Once the vector representations are constructed, the relevance score can be calculated by cosine similarity as defined in Equation 2.1.2.

$$rel(q,d) = \frac{< q,d >}{||q|| * ||d||} \tag{2.1.2}$$

where $q$ represents the query vector weighted with term frequencies and $d$ represents the document vector weighted with TF-IDF.

In order to further improve the performance of vector space model, many other weighting heuristics are proposed such as TF normalization which will decrease the term frequencies and not reward too much relevance score for terms with extremely high frequencies, and document length normalization, which will penalize long documents because they naturally have more words and tend to cover more words in query [Croft et al., 2009]. For example in the work of Singhal et al [Singhal et al., 1996], the authors designed a heuristic weighting scheme to incorporate document length normalization and the relevance score could be calculated as follows.

$$rel(q,d) = \sum_{w \in q \cap d} c(w,q)\frac{ln[1 + ln(1 + c(w,d))]}{1 - b + b \cdot \frac{|d|}{avdl}} \cdot log\frac{M+1}{df(w)} \tag{2.1.3}$$

where $c(w,q)$ represents the term frequency of $w$ in query $q$, $c(w,d)$ represent the term frequency of $w$ in document $d$, $|d|$ is the length of document $d$, $avdl$ is the average document

length of the document collection, $M$ is the total number of documents in the collection, $df(w)$ is the document frequency of term $w$, and $b \in [0,1]$ is a parameter of the model.

Although the VSM model is simple and straightforward, it still has some limitations. In VSM model, the terms are assumed to be independent in the vector space and each term in the vocabulary represent one dimension [Sun et al., 2004, Mabrouk et al., 2017], therefore the very simple VSM model will neglect the dependency between terms [Billhardt et al., 2002, Nallapati and Allan, 2002].

In order to overcome this shortcoming, the Generalized Vector Space Model is introduced [Wong et al., 1985], where the query and document representations are first mapped with a linear transformation before computing the scalar-product-based similarity score. This offers the possibility to relax the hard constraint of term independence and incorporate the relationships between terms into the linear transformation matrix. One popular way to implement this idea is to do a Latent Semantic Indexing (LSI) [Deerwester et al., 1990] which will basically conduct a SVD (Singular Value Decomposition) of the TF-IDF matrix and maps the terms into the latent space. In some experiments on small datasets, it has been shown that LSI can improve the performance of VSM. However, the creation of the latent space and the mapping of terms into it is done in order to minimize the Frobenius distance between the original matrix and the (truncated) transformed matrix, which may not necessarily project similar terms into the same dimension of the latent space. As a result, LSI failed to produce significant improvements on larger TREC collections [Atreya and Elkan, 2010].

Others studies such as Mitra et al [Mitra et al., 1997] and Fagan et al [Fagan, 1987] try to incorporate the compound terms, i.e. phrases into the the vector space. In those models, in addition to the term dimensions, the phrases are considered as additional dimensions in the vector space. Then the relevance score is calculated as a linear combination of the term relevance and phrases relevance as in Equation 2.1.4

$$rel(q,d) = w_{term} \cdot score_{term}(q,d) + w_{phrase} \cdot score_{phrase}(q,d) \tag{2.1.4}$$

where $w_{term}$ and $w_{phrase}$ are the weights of term score and phrase score respectively, which controls the importance of the term similarity score and phrase similarity score. The effectiveness of the approach is largely conditioned by the phrase dictionary, which may not cover all the meaningful phrases. In addition, it is also problematic to consider a phrase to be independent from its constituent words, as they are represented by separate dimensions in the vector.

One huge advantage of VSM model is its efficiency due to the building of 2 explicitly separated representation for query and document. It is possible to incorporate various features in both the query and document representations. However this advantage of having 2 separate representations for query and vector comes at the cost of having to design heuristics

for the weighting scheme manually [Zhai, 2008, Ko, 2012]. In the work of Zobel and Moffat [1998] it is observed that "No component or weight scheme was shown to be consistently valuable across all of the experimental domains". Despite this, the VSM is often used as a basic IR model due to its simplicity and reasonable effectiveness across test collections.

### 2.1.2. Probabilistic Model

In probabilistic model [de Campos et al., 2002], the relevance function is often associated with $P(R|d,q)$, i.e. the probability of relevant or not given the document and query, where $R$ is a random variable taking values from {0,1} representing relevant or not. In IR literature, many probabilistic models have been proposed which could be roughly divided into 2 categories: Binary Independence Model (BIR) [Manning et al., 2009] and language model (LM) [Croft et al., 2009].

2.1.2.1. Binary Independence Model. In binary independence model, we assume that the terms in queries and documents are independent and we represent their occurrence by binary vectors. For example $d = [1,0,0,1,1]$ signifies that the term $w_1, w_4, w_5$ occurred in document d, while $w_2, w_3$ do not. The goal is to estimate the ratio of $\frac{P(R|q,d)}{P(NR|q,d)}$ as the relevance score, where $R$ represents the relevance, $NR$ represents the non-relevance, $q$ represents the query and $d$ represents the document.

The direct estimation of the ratio $\frac{P(R|q,d)}{P(NR|q,d)}$ is difficult, however, we could employ Bayes Rule to inverse the random variables.

$$\frac{P(R|q,d)}{P(NR|q,d)} = \frac{\frac{P(R|q)P(d|R,q)}{P(d|q)}}{\frac{P(NR|q)P(d|NR,q)}{P(d|q)}} = \underbrace{\frac{P(R|q)}{P(NR|q)}}_{\text{independent of d}} \underbrace{\frac{P(d|R,q)}{P(d|NR,q)}}_{\text{need estimation}} \tag{2.1.5}$$

As we can observe from Equation 2.1.5, only the second factor depends on document, so we can estimate only the second factor for the purpose of document ranking. Since the terms are assumed to be independent, we can develop the formula into a product of individual probabilities.

$$\frac{P(d|R,q)}{P(d|NR,q)} = \prod_i \frac{P(d_i|R,q)}{P(d_i|NR,q)} = \prod_{d_i=1} \frac{P(d_i=1|R,q)}{P(d_i=1|NR,q)} \prod_{d_i=0} \frac{P(d_i=0|R,q)}{P(d_i=0|NR,q)} \tag{2.1.6}$$

Let $p_i = P(d_i = 1|R,q)$ and $s_i = P(d_i = 1|NR,q)$ and assume that for all terms not occurring in query, $p_i = s_i$, then, the above formula becomes

$$\prod_{d_i=1} \frac{P(d_i=1|R,q)}{P(d_i=1|NR,q)} \prod_{d_i=0} \frac{P(d_i=0|R,q)}{P(d_i=0|NR,q)} = \underbrace{\prod_{d_i=q_i=1} \frac{p_i(1-s_i)}{s_i(1-p_i)}}_{\text{dependent of documents}} \cdot \underbrace{\prod_{q_i=1} \frac{1-p_i}{1-s_i}}_{\text{independent of documents}} \tag{2.1.7}$$

Since the second factor is independent of the document, we only needs to estimate the first factor the Equation 2.1.7. Then, we take logarithm of the above formula and the relevance function would be

$$rel(q,d) = \sum_{d_i=q_i=1} \frac{p_i(1-s_i)}{s_i(1-p_i)} \tag{2.1.8}$$

In order to estimate this relevance function, we can draw a contingency table, in which each entry registers the number of documents in each case, as shown in Table 2.1. As such the

**Table 2.1.** Contingency Table of Term Occurrence

| | Relevant | Non-relevant | Total |
|---|---|---|---|
| $d_i$=1 | $r_i$ | $n_i$-$r_i$ | $n_i$ |
| $d_i$=0 | $R - r_i$ | $N$-$n_i$-$R$+$r_i$ | $N$-$n_i$ |
| Total | $R$ | $N - R$ | $N$ |

relevance function in Equation 2.1.8 could be estimated by

$$\sum_{d_i=q_i=1} log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i + r_i + 0.5)} \tag{2.1.9}$$

where the added 0.5 is a smoothing term to deal with 0 count.

The binary independence model also give inspiration to birth of the well known BM25 model [Robertson et al., 2004], which is an extension to the binary independence model to include term weights. This extension is based on both probabilistic arguments and empirically validated heuristics [Croft et al., 2009].

There are a lot of variants of the relevance function of BM25 model. One common form could be expressed as follows [Croft et al., 2009]:

$$\sum_{i \in Q} log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \cdot \frac{(k_1 + 1)fd_i}{K + fd_i} \cdot \frac{(k_2 + 1)fq_i}{k_2 + fq_i} \tag{2.1.10}$$

where $K$ is a parameter for document length normalization:

$$K = k_1((1 - b) + b\frac{dl}{avdl}) \tag{2.1.11}$$

where $r_i$, $R$, $n_i$, $N$ are counts defined in Table 2.1, $fd_i$ is the frequency of term $i$ in document, $fq_i$ is the frequency of term $i$ in query, $k_1$ and $k_2$ are parameters controlling how the weight of term $i$ will be discounted in document and query respectively, and $b$ is a parameter to control document length normalization.

In addition to binary independence model, there is another branch of probabilistic model which directly incorporates term frequency into the model based on statistical language model. We review this model in the next section.

2.1.2.2. Language Model. The approaches based on statistical language modeling estimate the relevance function by query likelihood [Croft et al., 2009], i.e. the probability of

query $q$ being generated from the document language model as follows.

$$rel(q,d) = P(q|d) \tag{2.1.12}$$

Usually we assume the independence between terms and employ unigram language model. Therefore the probability of $P(q|d)$ can be calculated as follows.

$$P(q|d) = \prod_{w \in q} P(w|d) \tag{2.1.13}$$

If we take logarithm of this probability the relevance function will become,

$$rel(q,d) = logP(q|d) = \sum_{w \in q} logP(w|d) = \sum_{w \in V} c(w,q) logP(w|d) \tag{2.1.14}$$

where $w$ is a term in vocabulary, $c(w,q)$ is the frequency of term $w$ in query. The probability of $P(w|d)$ could be estimated by maximum likelihood (ML) [Akaike, 1998]. However there might be unseen word in $d$. Thus smoothing is needed.

There are two popular smoothing schemes used in language models in IR, Jelinek-Mercer smoothing and Dirichlet smoothing [Croft et al., 2009]. In Jelinek-Mercer smoothing, the probability of $P(w|d)$ is estimated in an interpolated way.

$$P(w|d) = (1 - \lambda)\frac{c(w,d)}{|d|} + \lambda P(w|C),\ \lambda \in [0,1] \tag{2.1.15}$$

where $\lambda$ is the interpolating parameter controlling the importance of ML estimation and smoothed LM from document collection, $|d|$ is the document length of $d$. In Dirichlet smoothing, the probability of $P(w|d)$ is estimated as follows,

$$p(w|d) = \frac{|d|}{|d| + \mu} \cdot \frac{c(w,d)}{|d|} + \frac{\mu}{|d| + \mu} \cdot P(w|C) \tag{2.1.16}$$

where $|d|$ is the document length of $d$, and $\mu$ is the parameter of Dirichlet smoothing.

The relevance function of LM with Jelinek-Mercer and Dirichlet prior smoothing could be derived as follows.

$$rel_{JM}(q,d) = \sum_{w \in q \cap d} c(w,q) log[1 + \frac{1 - \lambda}{\lambda} \cdot \frac{c(w,d)}{|d|P(w|C)}],\ \lambda \in [0,1] \tag{2.1.17}$$

$$rel_{Dir}(q,d) = [\sum_{w \in q \cap d} c(w,q) log[1 + \frac{c(w,d)}{\mu P(w|C)}]] + nlog\frac{\mu}{\mu + |d|} \tag{2.1.18}$$

The classic LM based models in IR have demonstrated their effectiveness. However they have the limitation of only modeling the unigram language model. One can naturally extend the unigram model to bigram and trigram models. Song and Croft [1999] experimented those extensions.

We can observe that all the traditional IR models are defined manually, with the parameters fixed by empirical experiments. A legitimate question is whether human expert

would be able to define such a score function, given the fact that we only have very simple representations for documents and queries and we do not know exactly how relevance is judged by humans. In order to overcome the disadvantage of manually defining functions and parameters, learning-to-rank (L2R) models are proposed to learn a ranking function. In the next section, L2R models will be reviewed in detail.

### 2.1.3. Learning to Rank

Learning-to-rank models (L2R) try to learn a ranking function in a supervised manner to incorporate several features into the model, so as to determine the rank of a document for a given query. Once the model is trained, it can be applied to a new document-query pair [Liu, 2009]. The features used in L2R typically include two categories: those related to the document and to the query, and those related to their relationship. The first category of features includes the document and the query such as query length, document length, the sum of matched terms' tf-idf value etc., and webpage specific features such as PageRank score and HITS score [Bai et al., 2010, Macdonald et al., 2012]. The second category includes the basic relevance scores like BM25, LM of $(q,d)$ discussed in the previous sections.

According to the goal of the objective function and the type of labeled training data, the learning to rank approach could be roughly divided into 3 types [Liu, 2009]: pointwise approach, pairwise approach, and listwise approach.

In pointwise approach, the training instances is in a format of $\{(q,d), l\}$, given a query $q$, each document $d$ will be labeled with a degree of relevance $l$ which could be binary, integer or real scalar and the goal is to learn to estimate the degree of relevance. In pointwise approach, according to different types of labels (binary or real scalar), the problem is often regarded as a classification or regression problem, as such the classification loss or regression loss is employed.

In listwise approach, each training instance is of the form $\{[q,(d_1,d_2)], l\}$, i.e. a query and a pair of document as input and the label $l$ is the pairwise preference of the 2 documents which takes values from $\{+1, -1\}$, i.e. if $d_1$ is ranked higher than $d_2$, then, $l = +1$, otherwise $l = -1$. In pairwise approach, given a query $q$ the loss function measures the inconsistency between $h(d_1,d_2)$ and the true label $l_{d_1,d_2}$. Therefore many pairwise approach regards the problem as a classification problem and employs classification loss [Liu, 2009].

In listwise approach, each training instance is in the format of $\{q,[d_1,...,d_n],l\}$, i.e. a query and a list of documents, and the associated label $l$ is a ranked list of documents or its permutation. Since the input of the model is a query and a list of documents and the output should also be a ranked list of those documents, for practical reasons, the hypothesis of the model is often a combination of 2 functions, $h(q,\mathbf{x}) = sort \circ f(q,\mathbf{x})$ where $f$ is determined from the ranking preferences and *sort* is a function to sort according to the relevance. That

is, the model will first calculate a relevance score for each document with respect to the given query and then rank those documents according to their relevance scores [Liu, 2009].

Many machine learning methods can be used to implement the L2R approaches. For example for pointwise approach, if the labels are binary the model could be a SVM [Tong and Koller, 2001] classifier where the relevant documents are labeled as $+1$ and non relevant documents are labeled $-1$. If the labels are real scalars, the model might employ subset ranking [Cossock and Zhang, 2006] and treat the problem as a regression problem, and use the real-valued relevance as label.

Pointwise approach relies heavily on manually annotated relevance scores, which might be inconsistent because usually many annotators are involved. However their judgement on preference of one document over another is more consistent. Therefore pairwise comparison is more suitable for the goal of IR tasks which aims at ranking documents according to their relevance rather than assigning an absolute score. A lot of pairwise models are proposed, among which RankNet [Burges et al., 2005] is an interesting one.

2.1.3.1. RankNet. The RankNet model is a pairwise model which consists in building a multi-layer perceptron (MLP) as relevance function $f$, and the sigmoid function is employed as activation function for the hidden layer. Since it is a pairwise model, given 2 documents $u$, $v$ and their corresponding features $x_u$ and $x_v$, the label $l_{u,v}$ is defined as a Boolean variable, if $u$ is more relevant than $v$, $l_{u,v} = 1$, otherwise, $l_{u,v} = 0$. Therefore, given a query $q$ and 2 documents' feature vectors $x_u$ and $x_v$ the model will estimate their relevance to the query $f(x_u)$, $f(x_v)$ and then calculate the probability $O_{u,v}$ defined as the document $x_u$ is more relevant than document $x_v$, and this output probability is modeled by a logistic function as follows.

$$O_{u,v} = \frac{exp(f(x_u) - f(x_v))}{1 + exp(f(x_u) - f(x_v)} \tag{2.1.19}$$

where $f(x_u)$ and $f(x_v)$ are the relevance scores of document $u$ and $v$. The loss function is defined as the cross-entropy of the output, as shown in Equation 2.1.20.

$$L(x_u,x_v,l_{u,v}) = -l_{u,v} log O_{u,v} - (1 - l_{u,v}) log(1 - O_{u,v}) \tag{2.1.20}$$

where $l_{u,v}$ is the ground truth label of the training pair $(x_u,x_v)$, and $O_{u,v}$ is the output of the relevance model. Once the model is built, the training could be done with standard back-propagation [Hecht-Nielsen, 1988].

Later, Matveeva et al. [2006] extended the RankNet model by adding nested rankers on top of RankNet model in order to further improve the retrieval performance. The model will iteratively rerank top documents with RankNet algorithm, and found to be able to improve the performance on the top 10 documents.

2.1.3.2. LamdaRank. Although the RankNet model has demonstrated its strength, it still has some limitation. Since it is a pairwise model, and the loss function is defined to minimize the preference errors, i.e. situations where the true label $l_{u,v} = 0$ and the output probability $O_{u,v} \geq 0.5$ or $l_{u,v} = 1$ and $O_{u,v} < 0.5$. The model will tend to only minimize this pairwise error without taking into account the fact that the correct ranking of top documents is more important than those at lower positions. As a consequence, improving the objective function does not necessarily lead to a better evaluation measure. In fact, the position-related evaluation measures are commonly used in IR. NDCG (Normalized Discounted Cumulative Gain) [Yilmaz et al., 2008] is a common measure used in IR, especially in web search, which is defined in Equation 2.1.21.

$$NDCG@n = (rel_1 + \sum_{i=2}^{n} \frac{rel(i)}{log_2 i})/IDCG@n \tag{2.1.21}$$

where $rel_i$ is the relevance judgement for the document ranked at the $i_{th}$ position, $n$ is the cut-off position and IDCG@n is the ideal DCG@n score generated by the best possible ranking for this topic. The deeper a relevant document is placed in the rank list, the more its contribution will be discounted.

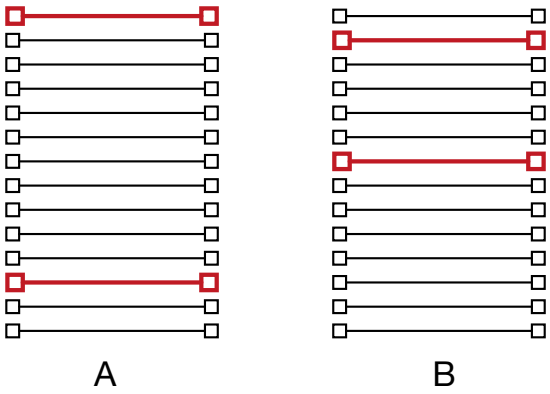



**Fig. 2.1.** Example of Limitation of RankNet

For example in Fig 2.1, each line represents a ranked document in the ranked list. Here we make a simple assumption that relevance scores are binary and take value from {0,1} and red line represents a relevant document, black line represents a non relevant document. It can be observed that in sub figure A, the pair error (number of pair mis-ranked where relevant document should be ranked higher than non relevant document) is 10. However if the RankNet model decide to descend the first document 1 position and move up the 12th document upwards to the 7th position, then the pair error will be reduced to 6. Although the pair error has been reduced, if we mesure NDCG@10, the acutal NDCG value is decreased

(because the negative impact on NDCG due to the first document's moving down is much larger than the positive impact of the 12th document's moving up).

In order to overcome this shortcoming, LambdaRank approach is proposed to imitate the idea of "gradient" which will push up relevant documents and descend non relevant documents according to a retrieval performance metric such as NDCG. This concept of "gradient" is called lambda function.

For example, in [Burges et al., 2006], the author optimizes on NDCG and the lambda function is derived as follows,

$$\lambda = Z_m \frac{2^{y_u} - 2^{y_v}}{1 + exp(f(x_u) - f(x_v))} (\frac{1}{log(1+i)} - \frac{1}{log(1+j)}) \tag{2.1.22}$$

where $Z_m$ is the reciprocal ideal DCG (IDCG) for a given query, $i$, $j$ are ranking positions for document $u$, $v$ ($u$ is more preferred than $v$) in previous iteration, $x_u$ and $x_v$ are input feature vectors and $y_u$ and $y_v$ are their corresponding labels. After each iteration the relevance scores of $u$ and $v$ will be updated by $+\lambda$ and $-\lambda$ respectively.

2.1.3.3. LambdaMART. In order to further improve the performance of the LambdaRank approach, a boosted decision tree based approach is proposed in [Wu et al., 2010]. This approach combines MART [Friedman, 2001] which is a boosted regression tree model with lambdaRank approach. The ensembled output of MART could be written as a linear combination of individual regression trees $f_i$.

$$F(x) = \sum_{i=1}^{N} \alpha_i f_i(x) \tag{2.1.23}$$

where $\alpha_i$ is the boosting parameter and $f_i$ is the individual regression tree model.

Considering the case where $n$ trees have been trained, in order to train the next tree, lambdaMART employs gradient descent to decrease the loss [Wu et al., 2010]. The lambda function could be derived as in [Burges, 2010]:

$$\lambda_{ij} = \frac{-\sigma |Z_{ij}|}{1 + exp(\sigma(s_i - s_j))} \tag{2.1.24}$$

where $\sigma$ is a parameter of the model to control the shape of the sigmoid function, $s_i$ and $s_j$ are the relevance scores for document $i$ and $j$, and $Z_{ij}$ is the utility difference caused by swapping the rank of $i$ and $j$, for example NDCG. If we choose to optimize the NDCG value, the updates for each leaf value $r_{km}$ in each tree $m$ is derived in [Burges, 2010].

$$r_{km} = \frac{-\sum_{x_i \in R_{km}} \sum_{i,j} |\Delta Z_{ij}| \rho_{ij}}{\sum_{x_i \in R_{km}} \sum_{ij} \sigma \rho_{ij} (1 - \rho_{ij})} \tag{2.1.25}$$

$\rho$ is defined as follows.

$$\rho_{ij} = \frac{1}{1 + exp(\sigma(s_i - s_j))} \tag{2.1.26}$$

where $\sigma$ is a parameter to control the shape of the sigmoid function.

Learning-to-rank approaches, which try to combine different features and implicitly learn a complex relevance function, have demonstrated their effectiveness in many applications. However, they still have some limitations among which a notable one is that the features they employ are still simple. As we mentioned, in many L2R models, the features employed are traditional relevance score like BM25 LM, some text statistics like the length of document title, document body and some web document specific scores like PageRank and HITS [Liu, 2009]. There isn't much high level representation involved in the learning process of L2R. If the user issues a query which asks about a concept or semantic item, the system might fail to satisfy the user's information need, because the manually defined features may fail to represent the concept.

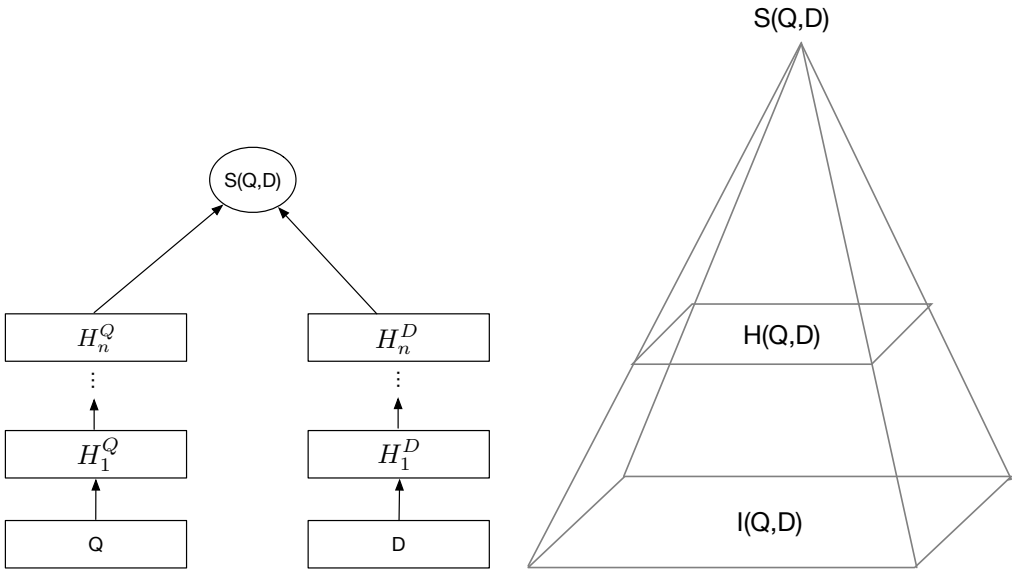
## 2.2. Deep Learning Model

Recently, with the rapid development of deep learning techniques, many attempts of applying deep learning techniques to IR have been made. One big advantage of deep learning techniques is that the model could learn a hierarchy of representations layer by layer from raw input, and automatically extract salient features and discovers patterns in raw input [Bengio, 2009, Bengio et al., 2012, 2013]. This avoids the inflexibility of hand-crafting features. Moreover, since the representations are learned layer by layer, at higher level, we can expect to have abstract and more semantic features extracted from raw text [Liang et al., 2017]. As such, deep learning model in IR adopted an approach which learns representations of query and document and apply a matching function to estimate relevance between query and document.

Depending on how the relevance is estimated, the neural IR models could be roughly divided into two families: representation- and interaction-based models [Guo et al., 2016]. The general structures of the two types of model are presented in Fig. 2.2. As we can observe from Fig. 2.2a Representation-based models first focus on learning meaningful representations through several hidden layers and then apply a similarity function on the last level query and document representations to estimate relevance.

Instead of learning semantic representation of query and documents, interaction-based models first compute local interactions between each query and document term at input and then learn the term-level interaction patterns through several hidden layers. The process is illustrated in Fig. 2.2b.

In this section, we will present some state-of-the-art representation-based models such as Deep Structured Semantic Model (DSSM) and Convolutional Deep Structured Semantic Model (CDSSM), and interaction-based models such as MatchPyramid and Deep Relevance Matching Model (DRMM).




**(a)** Representation-based Model **(b)** Interaction-based Model

**Fig. 2.2.** Representation- and Interaction-based Models

### 2.2.1. Deep Structured Semantic Model

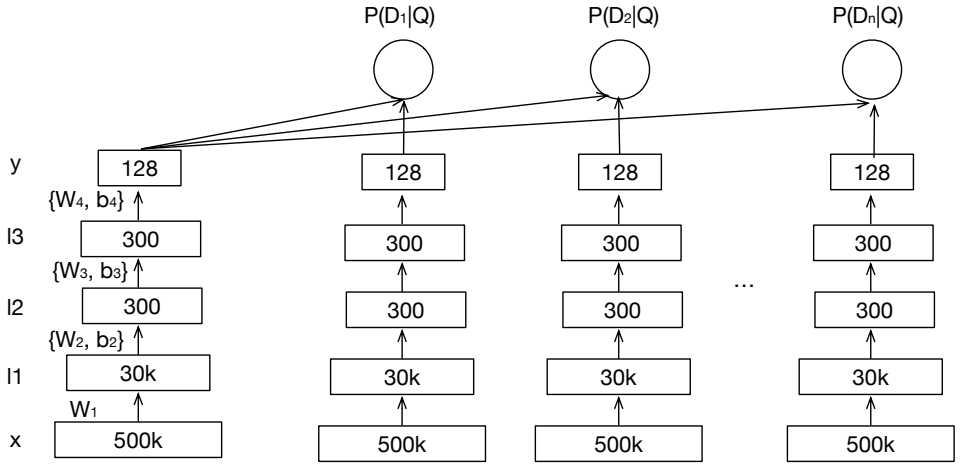



**Fig. 2.3.** DSSM Model

The introduction of Deep Structured Semantic Model (DSSM) [Huang et al., 2013] marked the first application of deep models in IR. The model builds 2 sets of deep feed-forward network to map the query and documents separately into representations layer by layer. At output, cosine similarity is employed to calculate the relevance of the document to the query, as shown in Fig 2.3.

In this model the query-side network has its own set of parameters $\{W_i^q, b_i^q\}$ and all document-side networks shares parameter $\{W_i^{doc}, b_i^{doc}\}$.

To deal with the large vocabulary, a letter n-gram representation of words is used. The query and documents are first converted into bag-of-word term frequency vectors in input layer $x$ and the dimension of $x$ is the vocabulary size, then the term frequency vector $x$ is mapped into a letter-ngram frequency layer by a fixed transformation matrix $W_1$.

$$l_1 = W_1 x \tag{2.2.1}$$

The authors call this process word hashing [Huang et al., 2013]. It works as follows: for each word, firstly, beginning and ending symbols are added, then the word are cut into letter-ngram, where $n$ is a hyper-parameter of the model. For example, in order to convert "good", beginning and ending symbols are added and the word becomes "#good#", if we choose $n = 3$ (i.e. letter-trigram), then it is converted to (#go, goo, ood, od#). The goal of conducting word hashing and map word frequency vectors to letter-ngram frequency vectors is two-fold. Firstly it greatly reduces the dimension of the input. In this work, the vocabulary size is around $500k$, after this mapping the letter-trigram dictionary size is reduced to about $30k$. This will alleviate the problem of curse of dimensionality and also reduce memory usage to hardware restrictions. Secondly, converting the words into letter-ngram avoids the out-of-vocabulary (OOV) problem. However an obvious risk is that this may create collisions, i.e. 2 different words having the same letter-ngram frequency vectors. However, after a thorough investigation with real-world data, the authors found out that by utilizing letter-trigram, there were only 22 collisions out of 30621 word tokens. Therefore the authors argue that the collision phenomenon could be ignored in front of the huge advantage of word hashing [Huang et al., 2013].

The conversion from term frequency vectors to letter-ngram frequency vectors is fixed and $W_1$ does't participate in training. After the input term frequency vector $x$ has been converted into letter-ngram frequency vectors $l_1$, it could be further forward propagated into higher layer by Equation 2.2.3. $tanh$ is employed as activation function in every layer.

$$l_i = f(W_i l_{i-1} + b_i),\ i = 2,...N-1 \tag{2.2.2}$$

$$y = f(W_N l_{N-1} + b_N) \tag{2.2.3}$$

The last layer $y$ contains highly abstract semantic information about raw input. Cosine similarity is employed to calculate the relevance between query and documents as follows:

$$R(Q,D) = cosine\_sim(y_Q, y_D) = \frac{y_Q^T y_D}{||y_Q||||y_D||} \tag{2.2.4}$$

At last there will be a softmax layer which will calculate the probability of the document $D_i$ being the relevant document $P(D_i|Q)$.

$$P(D_i|Q) = \frac{exp(R(Q,D_i))}{\sum_j exp(R(Q,D_j))} \tag{2.2.5}$$

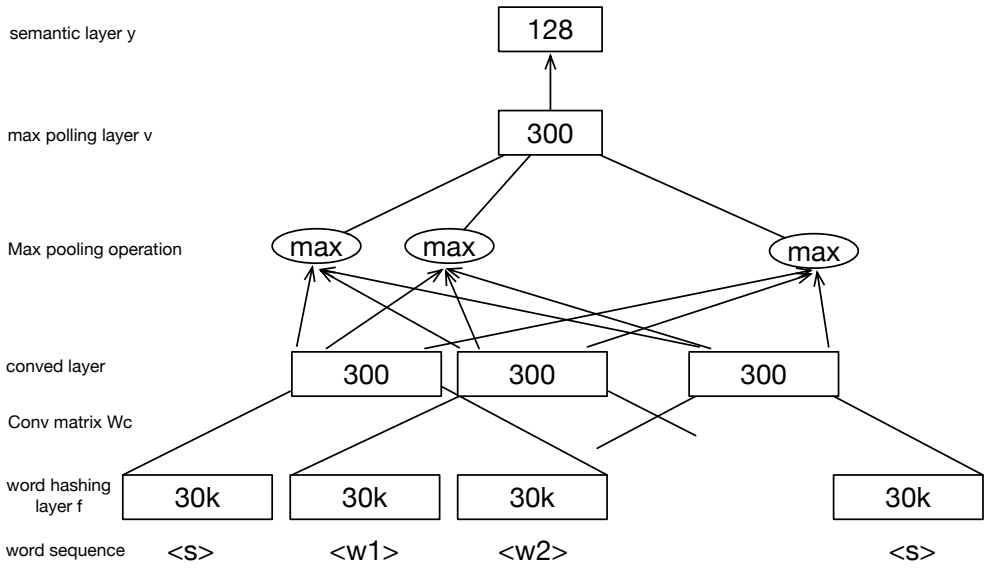



**Fig. 2.4.** CDSSM Model

The training is done in a supervised way: each training instance is of the format $(Q, D^+, D^-, ..., D^-)$, where $Q$ is the query, $D^+$ is a relevant document (positive example), $D^-$ is a non relevant document (negative example). The number of non relevant documents used in training is a hyperparameter. The model will minimize the following loss function.

$$L(\Lambda) = -log \prod_{Q,D^+} P(D^+|Q) \tag{2.2.6}$$

As a first attempt of applying complex deep structures to conduct IR tasks, the DSSM model has demonstrated its effectiveness by learning abstract representations of the raw text through several non-linear layers. The word hashing technique has also been employed to reduce dimensionality and avoid OOV problem. Despite its success, the DSSM still has its limitation since it uses bag-of-words representations at input level, which fails to take into account sequential information in original query and documents. Moreover, in its experiment settings, the authors only tried to match queries with document titles rather than the whole document, which contains much more non relevant terms. When DSSM is used on the whole document, the resulting document representation could be noisier, which may in turn hinder the effectiveness of DSSM.

### 2.2.2. Convolutional Deep Structured Semantic Model

In an attempt to exploit sequential information, Shen et al. [2014] proposed a convolutional deep structured semantic model (CDSSM) structure which aligns query and document words in a sequential order as input, and 1D convolution is used to map raw features into hidden layer.

Fig 2.4 shows a module to learn representation for a query or a document. At input layer, words in a query or a document are aligned in a sequential order. Then the same word hashing procedure used in DSSM is applied to each word to convert it into letter-ngrams. Once word hashing is completed, each vector in word hashing layer $f_t$ contains frequency count of each word in a sequential order. Then, the letter-ngram vectors of neighboring words in a context window of size $2d + 1$ are concatenated, and mapped into a vector of dimension $dim_{conved}$ (which is 300 in the author's setting) as follows.

$$h_t = tanh(W_c[f_{t-d}^T,...f_t^T,...f_{t+d}^T] + b_c) \tag{2.2.7}$$

where $h_t$ is the convolved hidden layer, $[\cdot,\cdot]$ represents concatenation, $f_{t-d}^T,...f_{t+d}^T$ represents the letter-ngram vectors in a context windows of size $2d + 1$, i.e. $d$ vectors before current word's letter-ngram vector $f_d^T$, and $d$ vectors after current word's letter-ngram vector, $W_c$, $b_c$ are the weight matrix and bias vector respectively, which are shared across all words, and $tanh$ is applied as activation function. As such, the convolution operation is done on a 1D space where the basic units are letter-ngram vectors $f_t$.

Then, the vectors in the convolved layer $h_t$ are subject to a max pooling operation which is done in a dimension-wise way across all convolved vectors. Suppose there are $m$ convolved vectors $h_t$ and each $h_t$ is of dimension $dim_{conv}$, the max pooling operation consists in extracting the max value from each of the dimensions of $h_t$ to form the max pooled vector $v$. The max pooled vector $v$ will have the same dimension $dim_{conv}$ as $h_t$. This max pooling operation could be formally described in Equation 2.2.8.

$$v(i) = \max_{j=1,...,T} h_t(i,j) \tag{2.2.8}$$

where $i$ represent the $i^{th}$ dimension of $h_t(i,j)$, $j$ represents one of the convolved vectors, and $T$ is the number of total convolved vectors. The max pooling operation is employed because it is believed to be able to suppress the non-significant local features and only retain the most salient features that are useful in IR tasks [Shen et al., 2014]. Once the max pooled layer $v$ is calculated, it is further mapped into a semantic layer $y$ which is a high level representation of the original input containing semantic information.

This module is applied to query and candidate documents separately with all candidate documents sharing the same parameter. After the semantic representations $y$ have been learned for both query $q$ and candidate document $d$, the relevance is calculated by a traditional cosine similarity as follows.

$$R(Q,D) = cosine(y_Q, y_D) = \frac{y_Q^T y_D}{||y_Q||||y_D||} \tag{2.2.9}$$

A softmax layer will be applied on top of the relevance vector to calculate the probability of the document $D_i$ being the relevant document $P(D_i|Q)$. The training is done in the same

way as DSSM, each training instance is a query $Q$ with a list of positive document (relevant) and negative documents (non relevant) $[D^+, D^-,...,D^-]$. The goal is to minimize the same loss function as in DSSM:

$$L(\Lambda) = -log \prod_{Q,D^+} P(D^+|Q) \tag{2.2.10}$$

The CDSSM has demonstrated its improvement over DSSM thanks to its convolutional structure which preserves some sequential information for input query and document. The authors conducted experiments on an clickthrough dataset to retrieve document titles, and got a 3.7% improvement on nDCG@10 over the DSSM model [Shen et al., 2014].

### 2.2.3. Match Pyramid

In addition to representation-based models such as DSSM and CDSSM discussed in previous section, there have been another family of neural IR models proposed in literature. Those models estimate the relevance of a query and document based on their interactions. Instead of first learning representations of query and document, they usually first build local term-to-term interactions between query and document to form an interaction pattern. Afterwards a series of neural layers are employed to analyze the interaction patterns. Finally, the last layer of the learned interaction information will be employed to output a relevance score.

Among those interaction-based models, a well-known one is the MatchPyramid [Pang et al., 2016a]. Its general architecture is presented in Fig 2.5.

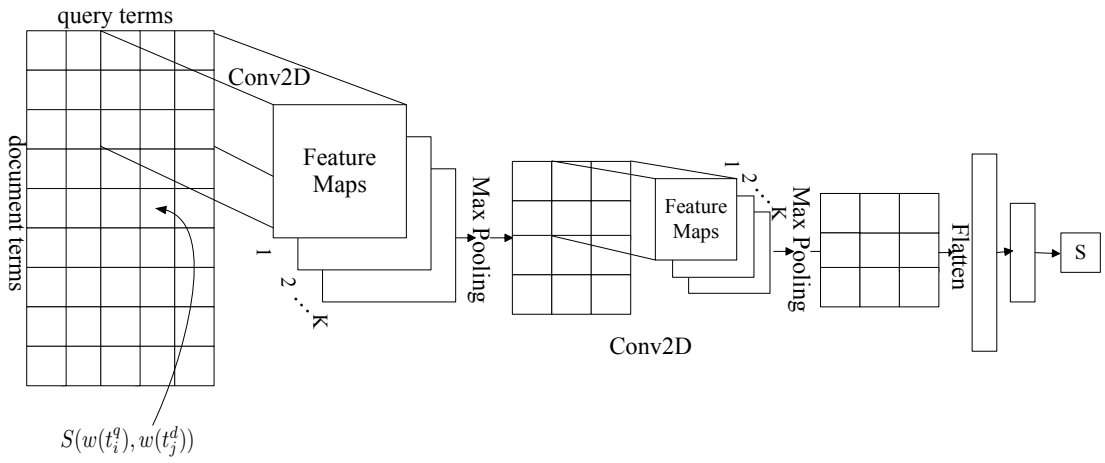



**Fig. 2.5.** Match Pyramid

In this model, the input feature is a 2D interaction matrix $I$ of dimension $|q| \times |d|$, where $|q|$ and $|d|$ represent the query and document length respectively. Each element $I_{ij}$ of the matrix $I$ is calculated as the cosine similarity of the representation $i^{th}$ query term and

the $j^{th}$ document term. Here the representations of terms could be pre-trained by off-the-shelf embedding tools such as Word2Vec [Goldberg and Levy, 2014]. The process could be summarized in Equation 2.2.11.

$$I_{ij} = cos(t_i^{(q)}, t_j^{(d)}) \tag{2.2.11}$$

Afterwards, the input interaction matrix $I$ will be fed into a series of 2D convolution and max-pooling layers and the last max-pooled layer will be flattened to form a 1D vector to represent the final pattern vector learned through the model. Finally this pattern vector will be fed into a dense layer to produce a relevance score. The whole process could be summarized as follows.

$$P_0 = max\_pool(I) \tag{2.2.12}$$

$$C_1^k = f(W_1^k * I + b_1^k),\ k = 1,..,K \tag{2.2.13}$$

$$P_1^k = max\_pool(C_1^k),\ k = 1,..,K \tag{2.2.14}$$

$$C_i^k = f(W_i^k * P_{i-1} + b_i^k),\ i = 2,..,L,\ k = 1,..,K \tag{2.2.15}$$

$$P_i^k = max\_pool(C_i^k),\ i = 2,..,L,\ k = 1,..,K \tag{2.2.16}$$

$$H = g(W^h P_K + b^h) \tag{2.2.17}$$

$$S = h(W^s H + b^s) \tag{2.2.18}$$

where $C_i^k$ is the feature map $k$ of the $i^{th}$ convolved layer; $I$ is the input interaction matrix; $W_i^k$ and $b_i^k$ are the kernel and bias of layer $i$ for the feature map $k$; $L$ is the number of convolution layers, and $K$ is the number of feature maps; $f$, $g$, $h$ are non-linear mappings; and $*$ represents the convolution operator.

The training is then done by minimizing a pair-wise loss function. Given a training example ($Q$,$D_+$,$D_-$), we hope that the positive score $S(Q, D_+)$ should be higher than the negative score $S(Q,D_-)$. The loss is defined in Equation 2.2.19, where $\Theta$ includes all trainable parameters in the model:

$$L(Q,D_+,D_-;\Theta) = max(0, 1 - (S(Q, D_+) - S(Q,D_-))) \tag{2.2.19}$$

### 2.2.4. Deep Relevance Matching Model

In DSSM, CDSSM, and Embedding-based Convolutional Model, only the semantic representations of the last layer are used in relevance estimation. However the low level interactions between query and document are discarded. To solve this problem, Guo et al. [Guo et al., 2016] proposed a Deep Relevance Matching Model (DRMM) which attempts to solve this problem. In this approach, local interactions at term level are employed in relevance estimation. The structure of the model is presented in Fig 2.6.

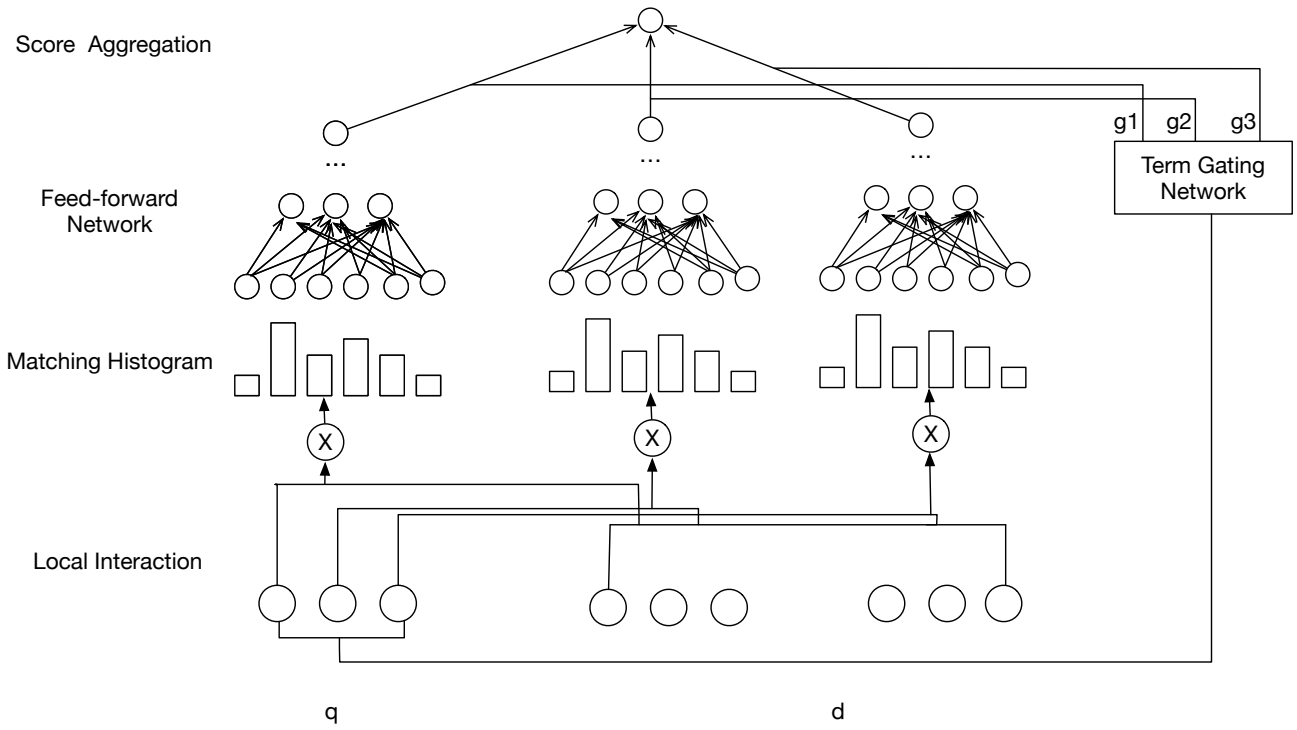



**Fig. 2.6.** DRMM

First, at input layer the word embeddings learned by off-the-shelf tools like word2vec [Mikolov et al., 2013] are employed. Then, interactions between query and document at term level are immediately calculated. The authors argue that if user's information need is an exact match, low-level interactions on terms are crucial. Since query and document could be of arbitrary length, in order to deal with the variable length problem, the DRMM model does't use traditional padding techniques, instead it employs an intensity-based histogram method to calculate the low-level interactions. For each query term's embedding $w_i$, the model will first calculate the scalar product of the query term $w_i$ with every document term's embedding $d_i$. Then, the interactions will be classified into several intervals. Finally the histogram will contain counts of interactions in each interval.

$$z_i^{(0)} = h(w_i \otimes d) \tag{2.2.20}$$

where $z_i^{(0)}$ is the interaction histogram vector for query term $i$, $w_i$ is the embedding of query term $i$, $d$ is a list containing embeddings of each document term, and $\otimes$ represents scalar product.

For each query term, the interaction histogram vector will be fed into a deep feed-forward network, the operation in each layer could be represented in Equation 2.2.21

$$z_i^{(l)} = tanh(W^{(l)} z^{(l-1)} + b^{(l)}) \;\; l = 1,...,L \tag{2.2.21}$$

where $z_i^{(l)}$ is the activation of layer $l$ for query $i$'s network, $W^{(l)}$ and $b^{(l)}$ are weight and bias of layer $l$ and are shared across all query term networks.

Finally the output score $Z_i^{(L)}$ from each query term's network will be aggregated in a weighted scheme

$$s = \sum_i g_i z_i^{(L)} \tag{2.2.22}$$

where $g_i$ is the aggregation weight for term $i$, and is calculated with a softmax function as follows.

$$g_i = \frac{exp(w_g x_i)}{\sum_j exp(w_g x_j)} \tag{2.2.23}$$

where $w_g$ is the weight of the term weight of the term gating network, and $x_i$ is the input of query term $i$ in the term gating network, which could be the embedding vector of term $i$ or the inverse document frequency of term $i$, depending on different experiment configurations [Guo et al., 2016]. The weight of this term gating network $w_g$ is set to be a learnable parameter and learned together with other model parameters.

The model is trained in a pairwise manner with hinge loss: given a training pair (q, d+, d-) where $q$ is a query, $d+$ is a document ranked higher than $d-$, the loss function could be expressed as

$$Loss(q,d+,d-) = max(0, 1 - s(q,d+) + s(q,d-)) \tag{2.2.24}$$

where $s$ represents the predicted relevance score of query $q$ and document $d$.

The experimental results reveal that DRMM outperforms representation-based models such as DSSM and CDSSM significantly on MAP and nDCGs [Guo et al., 2016]. The authors believe that for the task of adhoc retrieval, the most important signals are relevance matchings signals rather than semantic representations. Since DRMM as an interaction-based models focuses on analyzing the interaction pattern (i.e. the histogram) to estimate the relevance score, it provides some advantage over representation-based models, namely, it can capture fine-grained matching signals between document and query, which a representation-based model could have difficulties to achieve.

### 2.2.5. Pre-trained Language Model

Recently the rapid development of pre-trained language models (PLMs) have shown their potentials. Among those pre-trained language models, BERT [Devlin et al., 2019] is a popular one and has been applied to many NLP tasks thanks to its effectiveness and versatility. One of the advantages of the BERT model is that it pre-trains multiple layers of contextualized representations on large training corpus and could be adapted to perform different downstream tasks by fine-tuning it with the task specific layers or objective function.

There have been some work which employed BERT as representation learning layers for Information Retrieval tasks. For instance, TwinBERT [Lu et al., 2020a], employs two symmetrical BERTs to learn the query and document representations and the last layer's [CLS] token representations for query and document are employed as global representations to perform a match by dot product. The ColBERT [Khattab and Zaharia, 2020] also learns query and document representations separately by two BERT encoders. Afterwards, each query term representation of the last layer is matched with each document term representation of the last layer, and a max interaction value is retained. Finally the max interactions are aggregated for all query terms. By delaying the interactions until the completion of query/-document representation construction, it could achieve the desired fast response for serving queries.

## 2.3. Summary

In this chapter, both traditional models and the state-of-the-art neural models for IR are presented. From the above presentation, we can observe that a neural IR model essentially boils down to two parts: representation learning module and matching function. Depending on how the two parts are combined, the existing neural models could be categorized into representation-based models and interaction-based models. Representation-based models such as DSSM and CDSSM employ complex representation learning modules to learn a series of layers of representations and employ the last layer of query and document representation to perform a match. Usually this matching function is a simple one like cosine similarity. On the other hand, interaction-based models such as MatchPyramid and DRMM employ simple term representations such as term embeddings and build local term-to-term interaction such as interaction matrix or histogram at an early stage. Afterwards, a series of neural layers in charge of analyzing the matching patters are applied on the input pattern. Those layers are in charge of learning the global matching function. Finally the learned matching pattern in the last layer is employed to output a relevance score.

From the above description, it is not difficult to observe several limitations of the existing neural IR models. Firstly, regardless of representation- or interaction-based models, most of them only employs the representation or interaction pattern of the last layer to calculate a relevance score. As mentioned previously, deep neural models have the capability to learn more and more abstract information from input by a series of layers. The input layer is in charge of capturing information on term level and higher layers are believed to capture semantics on conceptual level. Hence those approaches which employ only the representation or interaction of the last layer to calculate a relevance score are more suitable for conceptual queries. However as discussed in [Guo et al., 2016] [Gysel et al., 2016], there are also lexical queries requiring an exact match. For example, the query "Ronald Reagan tax cut" asks

the system to retrieve document containing the tax cut policy of the former US president Ronald Reagan, any generalization of these terms to other president are not desirable. There could also be many cases in between. Therefore it is more appropriate to incorporate the representations or interactions of multiple layers into the process of estimating the relevance score in order to serve different types of query.

Secondly, matching between query and document shouldn't only happen on the same abstraction level. There's also the need of performing matchings across different layers of abstraction to satisfy the need of term-phrase matching. Very often we observe that in the document, a phrase could also be related to a single query term, and vice versa. For example, for the query term "mask", the phrase "personal protection equipment" in document is also relevant to it, since mask belongs to the concept of personal protection equipment. The hierarchy of layers of representations learned by neural models gives the possibility to achieve term-phrase matching, therefore it is also worthwhile to match representations of query and document from different layers to better satisfy this need.

Thirdly, the traditional L2R framework has demonstrated its potentials in learning a good ranking function. However it is observed that in many existing L2R models, the input features are fixed, hand-crafted features such as query and document length, IDF of query terms, BM25 and LM scores between query and documents. Those hand-crafted traditional features are very practical, however they lack the expressiveness of neural features learned by representation or interaction modules. On the other hand, existing neural IR models are mostly trained with a simple pairwise hinge loss, which are less effective than that of L2R framework. Moreover, It has often been observed that neural features and traditional features are complementary, and they are often combined to help improve performance [Kim et al., 2012, Severyn and Moschitti, 2015]. Therefore it is natural to raise the question: Is it beneficial to combine both the L2R framework and neural feature learning modules together to learn features and ranking function simultaneously in an end-to-end manner? If this is possible, the integrated learning model could combine both neural features and traditional features and learn the ranking function and neural features together to take advantage of both neural models and L2R framework.

In this thesis, we will address the limitations mentioned above.

# Chapter 3

# Collections and Evaluation Methods

In this section, we will present the evaluation framework used in this thesis. Section 3.1 describes the collections we used in our studies, Section 3.2 presents the external resources that we used in this thesis, and Section 3.3 presents the evaluation metrics employed in different studies.

## 3.1. Collections

In this study we employs several different public collections to conduct our experiments. They are the ClueWeb09B, MQ2007, MQ2008 and MSMARCO collections. The details of those collections are described in the following sections.

**ClueWeb09B**: The Text Retrieval Conference (TREC) [2] organized the web track 2009-2012 competitions and prepared this ClueWeb09 Collection. This collection contains webpages crawled from the Internet in January and February 2009. The size of this collection is very large, and we employ a commonly used subset of this collection called ClueWeb09B. The details of this collection is presented in Table 3.1.

**Table 3.1.** ClueWeb09B Statictics

| Collection | Genre | Validation Queries | Test Queries | #Docs | avg d length |
|---|---|---|---|---|---|
| Clueweb09B | Webpages | #1-50 | #51-200 | 50,220,423 | 1,506 |

The collection contains 50,220,423 documents. The genre of this collection is webpage, and the average document length is 1,506 terms which is relatively long.

The judgments for validation and test queries are provided with this collection by TREC webtrack 09-12 competitions. The judgments for documents this collection are graded from $-2$ to 4. $-2$ signifies that this document is a spam document it doesn't appear to be useful for any reasonable purpose. 0 signifies that this document is not relevant to the current

[2] http://trec.nist.gov/

topic, but may be relevant to other topics. 1 means that the content of this document is relevant to the current topic. 2 means that the content of this document is highly relevant to the topic. 3 signifies that this document is dedicated to the current topic, it is authoritative and comprehensive. 4 means that this document is the homepage of an entity directly named by the current query. An example of the judgments are presented in Fig 3.1,

```
1 0 clueweb09-en0010-39-02185 0
1 0 clueweb09-en0009-84-37392 0
1 0 clueweb09-enwp01-17-09993 2
1 0 clueweb09-enwp00-61-20162 2
1 0 clueweb09-enwp00-27-06080 1
1 0 clueweb09-en0008-03-02179 0
1 0 clueweb09-en0004-34-30809 0
1 0 clueweb09-enwp01-58-04573 0
1 0 clueweb09-en0009-30-02765 0
1 0 clueweb09-enwp03-13-02851 0
1 0 clueweb09-enwp00-37-05605 1
1 0 clueweb09-enwp00-18-04573 0
```

**Fig. 3.1.** An Example of ClueWeb09 Judgment

where each line represent a judged query-document pair. The first column represents the query id, the second column is a placeholder, the third column is the document id, the forth column is the judgment.

**MSMARCO**: The MSMARCO reranking dataset [1] is released by Microsoft. The queries are real user issued questions extracted from Bing [2] query log, and the documents are candidate answer passages. This dataset already provided a baserun by BM25 and the goal of this dataset is to rerank the passages given a query. The statistics of this dataset are presented in Table 3.2. The size of this collection is large. It contains 8,841,823 documents and 1,010,916 queries. Since the genre of the collection is answer passages, the average document length is only 58 terms, which is relatively short. In this dataset, in order to facilitate training, the authors of the dataset have already provided training triples (query, document, label) in 2 format: small (27.1GB) and large (272.2GB). To facilitate our experiments, we chose to use the small set of training triples.

As for evaluation, the authors of the dataset have already prepared a baseline run based on BM25, and provided the development and test query-document pairs. Therefore we follow the testing protocol and rerank the provided baseline run by our proposed models.

[1] https://microsoft.github.io/msmarco/
[2] https://www.bing.com

**Table 3.2.** Statistics of MSMARCO Dataset

| MSMARCO | |
|---|---|
| # Docs | 8,841,823 |
| # Queries | 1,010,916 |
| # Training triples (small) | 39,782,779 |
| # Training queries | 68,750 |
| # Validation queries | 698 |
| # Test queries | 6,282 |
| # Avg doc len | 58 |

The judgments are graded in a binary manner, i.e. if a document is relevant to a query, it will be judged as 1, otherwise, it will be judged as 0. Note that since this dataset is for answer passage reranking task, there's only one relevant passage per query, which is the correct answer to the query. An example of the judgments are presented in Fig 3.2, where

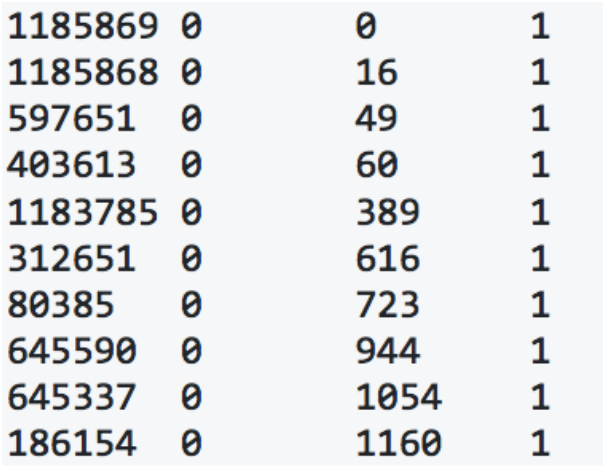



**Fig. 3.2.** An Example of MSMARCO Judgment

each line represents a judged query-passage pair. The first column represents the query id, the second column is a placeholder, the third column is the document id, the forth column is the judgment.

This collection is used in Chapter 5, in which we investigate if allowing cross-matches between term and phrase representations will help to boost performance.

**MQ2007 and MQ2008**: The MQ2007 and MQ2008 datasets [1] contain documents from the Gov2 collection [2] provided by TREC. The documents are crawled from gov websites in early 2004 and the average document length is 956. Both datasets contain human assessor's judgment triples (query, document, judgment) and the datasets have already been split into

[1]https://www.microsoft.com/en-us/research/project/letor-learning-rank-information-retrieval/
[2]https://www-nlpir.nist.gov/projects/terabyte/#data

5 folds. For our experiments, we stick to the orignal splitting of the 5 folds to perform 5-fold cross-validation. The details of the two datasets are presented in Table 3.3. The MQ2007

**Table 3.3.** Statistics of the MQ datasets

| | #queries | #docs | #rel_q | #rel_per_q |
|---|---|---|---|---|
| MQ2007 | 1,692 | 65,323 | 1,455 | 10.3 |
| MQ2008 | 784 | 14,384 | 564 | 3.7 |

dataset consists of 5 folds, which contains in total 1692 queries and 65,323 documents, the number of queries with at least one relevant document is 1455 and the average number of relevant document per query is 10.3. The MQ2008 dataset is smaller than the MQ2007. It also consists of 5 folds, which has in total 784 queries and 14,384 documents, the number of queries with at least one relevant document is 564 and the average number of relevant document per query is 3.7. Because the size of MQ2008 is small, it might not be enough to train a large neural model, following the practice of [Fan et al., 2018, Pang et al., 2017], we merge the training folds of MQ2007 into MQ2008 and keep the validation and test fold unchanged.

As for evaluation, since those datasets have already been split into 5 folds by the author of the datasets, we perform 5-fold cross validation, using 3 folds for training, 1 fold for validation and the remaining fold for testing. Following the practice of [Fan et al., 2018, Pang et al., 2017], during evaluation, we do not rerank the validation and test instances over a baseline run, but directly rank the validation and test query-document pairs provided in the respective folds.

The judgment labels in these datasets are graded from 0 to 2. 0 signifies that the current document is not relevant to the topic, 1 means that the document is relevant to the query and 2 indicates that the document is highly relevant to the topic. An example of the judgments for these datasets are presented in Fig 3.3 where each line represent a judged query-document pair and their judgment. The first column represents the query id, the second column represents the document id, the third column is the judgment.

These two datasets are very classic and have been employed in a series of traditional IR models that doesn't involve learning and some traditional, non-neural feature-based L2R models. However the major drawback of these datasets is that they are quite small (1692 and 784 queries) compared with MSMARCO (68,750 queries) and ClueWeb+AOL (8,969,337 queries) in terms of the size of training data. As discussed above, to train a deep model, it is preferable to have a large amount of training data. In experiment, we did find that for these two datasets, sometimes, in additional to the neural features learned by our model, we need to add traditional non-neural features contained in these datasets to boost performance. We employ these two datasets in the studies presented in Chapter 5 and 6.

```
10      GX057-59-4044939        1
10      GX057-97-13260700       0
10      GX058-98-1548149        0
10      GX068-48-12934837       1
10      GX169-47-11414154       0
10      GX197-89-11306652       0
10      GX225-79-9870332        1
10      GX235-13-6960344        0
10      GX235-84-0891544        1
10      GX236-05-0163071        1
10      GX236-56-7623480        1
```

**Fig. 3.3.** An Example of MQ Judgment

In summary, we have 4 collections presented above. The details of those collections are summarized in Table 3.4.

**Table 3.4.** Summary of Collection Statistics

| Collection | #Queries | #Docs | #Avg Doc Length | Genre | Judgement | Test Scheme |
|---|---|---|---|---|---|---|
| ClueWeb09B | 200 | 50,220,423 | 1,506 | Webpages | Graded | Reranking |
| MSMARCO | 1,010,916 | 8,841,823 | 58 | Answer passages | Binary | Reranking |
| MQ2007 | 1692 | 65,323 | 956 | Gov doc pages | Graded | Ranking |
| MQ2008 | 784 | 14,384 | 956 | Gov doc pages | Graded | Ranking |

The ClueWeb09B collection is a very big collection, containing webpages crawled from the web in early 2009. The documents in this collection are also very long, consisting of thousands of terms. The judgement for this collection are labeled in a graded way, and we rerank the BM25 baseline results to test our model's performance.

The MSMARCO is also a very large collection. This collection contains questions and answer passages. Since the documents are answer passages, they are usually short in length. This allows easier learning of the document representations. The judgement for this collection is labeled in a binary way, and we perform reranking on a BM25 baseline to test our model's performance.

MQ2007 and MQ2008 are two relatively small datasets which contains several hundreds to more than one thousand queries. The documents in this collections are government document pages of hundreds of terms. The judgements are labeled in a graded way, and following [Pang et al., 2017], we perform 5 fold cross validation and directly rank the test queries in each test fold, rather than reranking a baseline result.

## 3.2. External Resources

**GloVe Embeddings**: To better represent the term in neural IR models, we employ the GloVe word embeddings [1] learned by [Pennington et al., 2014] to initialize the embeddings of query and document terms. The set of embeddings we employed are trained on wikipedia dump (2014) with 6 billion tokens, the vocabulary size is kept as 400K, and the dimension of the embedding vectors is 300.

The advantage of using GloVe embeddings over the word2vec [Mikolov et al., 2013] is that the GloVe embeddings considers global co-occurence of terms rather than the terms in the local context window in word2vec. Experiment results on the task of ranking similar word pairs [Agirre et al., 2009] demonstrated its superiority over word2vec embeddings [Pennington et al., 2014].

**AOL query log**: The training of a deep IR model requires a large amount of labeled training data. In the lack of a large amount of training data, the recent proposed idea of weak supervision [Dehghani et al., 2017] framework could be employed to generate weakly supervised training data. The general idea of weak supervision is to select a set of queries and launch the queries with a traditional, non learning-based retrieval model such as BM25 against a document collection to retrieve documents and their scores, afterwards the scores could be regarded as weak label. The details of weak supervision will be presented in Chapter 4. For the work "Multi-level Abstraction Convolution Model" presented in Chapter 4, in order to generate weakly supervised query-document pairs, we employed the AOL query log [Pass et al., 2006] as training query set.

The AOL query log [2] contains 32,777,610 anonymized real user queries issued by 657,426 users from March 1st till May 31, 2006. Each entry of the query log contains an anonymized userID, the query text, date, time of submission, the position of the clicked document and the document URL. The structure of the query log and some typical queries are presented in Table 3.5.

**Table 3.5.** Typical Queries of Aol Query Log

| userID | query | date | time | click position | URL |
|---|---|---|---|---|---|
| 142 | dfdf | 3/24/06 | 22:23:07 | | |
| 142 | merit release appearance | 4/22/06 | 23:51:18 | | |
| 217 | susheme | 3/2/06 | 12:31:08 | | |
| 217 | lottery | 3/1/06 | 11:58:51 | 1 | http://www.calottery.com |
| 1337 | kentucky fried chicken | 4/25/06 | 16:07:14 | 1 | http://www.kfc.com |
| 2178 | bare minerals make up | 4/7/06 | 15:36:02 | 3 | http://www.essentialdayspa.com |
| 2334 | jojo lyrics | 3/19/06 | 15:12:26 | 1 | http://www.azlyrics.com |

[1] https://nlp.stanford.edu/projects/glove/
[2] http://octopus.inf.utfsm.cl/ juan/datasets/

For the purpose of generating training query-document pairs, we only employs the query text and discard the other information. We filter out navigational queries (Queries containing URL strings "www.", ".com", ".org", ".net", ".edu") and queries containing non-alphanumeric characters as done in [Dehghani et al., 2017]. This results in 8,969,337 training queries.

## 3.3. Evaluation Metrics

The goal of information retrieval is to return a ranked list of document give a query issued by user. Among the retrieved documents in the list, there could be some documents judged positive and others judged negative. The evaluation metrics will evaluate how well the system retrieves relevant documents from the collection and ranks them at top positions in the list.

### 3.3.1. Binary Metrics

**P@k**: The precision at k [Zuva and Zuva, 2012] calculates the precision at the cut-off position $k$ in the ranked list. It is calculated as follows:

$$P@k = \sum_{i=1}^{k} \frac{rel(d_i)}{k} \tag{3.3.1}$$

where $k$ is the cut-off position in the ranked list, $rel(d_i)$ is the judgment of the $i^{th}$ document in the ranked list, and $rel(d_i) \in \{0, 1\}$.

**AP**: The average precision considers the precision at different positions in the ranked list where a relevant document is placed. It is defined as follows.

$$MAP = \sum_{i:rel(d_i)=1} \frac{P@i}{|R|} \tag{3.3.2}$$

where $rel(d_i)(\in \{0, 1\})$ is the judgment of the $i^{th}$ document in the ranked list, $rel(d_i) = 1$ means a relevant document, and $|R|$ is the number of relevant documents.

**MAP**: The mean average precision [Craswell, 2009b] calculates the average precisions (AP) over all test topics (queries). It is defined as follows.

$$MAP = \sum_{q \in Q} \frac{AP(q)}{|Q|} \tag{3.3.3}$$

where $q$ is a topic in the test topic collection $Q$, and $|Q|$ is the total number of test topics.

**MRR**: The mean reciprocal rank [Craswell, 2009a] measures how close to the top the relevant documents are ranked in the retrieved list. It is simple the mean of the reciprocal ranks of relevant document:

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{i_q} \tag{3.3.4}$$

where $i_q$ is the position of the relevant document in the retrieved list for topic $q$, and $|Q|$ is the total number of test topics. MRR is often used in the case where there is only one relevant document for a topic.

### 3.3.2. Graded Metrics

Graded metrics consider graded relevance judgments. For example, relevance judgments could be 0, 1, 2, 3 and 4, with 4 being perfectly relevant and 0 being non relevant. The most commonly used metric is NDCG.

**NDCG@k**: The Normalized Discounted Cumulative Gain at k is proposed in [Järvelin and Kekäläinen, 2002, Robertson, 2000]. The idea of proposing NDCG is to allow using graded judgments and penalize relevant documents which are ranked at lower positions. Firstly, the discounted cumulative gain is calculated as the gain for each document discounted by the logarithm of its ranked position as follows.

$$DCG@k = \sum_{i=1}^{k} \frac{2^{rel(d_i)} - 1}{log_2(i+1)} \tag{3.3.5}$$

where $k$ represents the cut-off position, $i$ is the current rank position, and $rel(d_i)$ is the relevance judgment of document $d_i$.

Once the discounted cumulative gain is calculated, the ideal discounted cumulative gain (IDCG@k) is calculated as the DCG@k of the best ranking order (by placing the most relevant ones on top). Finally the NDCG@k is obtained by normalizing DCG@k over IDCG@k as follows.

$$NDCG@k = \frac{DCG@k}{IDCG@k} \tag{3.3.6}$$

## 3.4. Summary

In this chapter, we presented the external resources that we employed in the studies of this thesis. We employ the pre-trained GloVe embedding vectors to initialize the term representations in our model to allow the models to have a better initial state in the parameter space. We also employ the queries in the AOL queyr log to generate weakly supervised training triples by the weak supervision framework for the study which will be presented in Chapter 4.

We then presented the collections we employed in this thesis, their genres and basic statistics, and the corresponding judgments. Finally we presented the evaluation metrics we used in this thesis, including binary metrics such as MAP and MRR, graded metrics like NDCG@k. We employ MAP and NDCG@k (k=1, 3, 10, 20) in the study presented in Chapter 4, 5 and 6 and MRR in Chapter 5.

# Chapter 4

# Multi-level Abstraction Convolution Model

## Article Details:

1. Yifan Nie, Alessandro Sordoni, Jian-Yun Nie: Multi-level Abstraction Convolutional Model with Weak Supervision for Information Retrieval. SIGIR 2018: 985-988

2. Yifan Nie, Yanling Li, Jian-Yun Nie: Empirical Study of Multi-level Convolution Models for IR Based on Representations and Interactions. ICTIR 2018: 59-66

## Context:

At the time of writing these articles, most of the existing neural IR models build either representations or interactions and only employs the last layer of representations or interaction patterns to perform matching. For example, for the representation-based models DSSM [Huang et al., 2013] and ARC-I [Hu et al., 2014], only the last layer of learned query and document representations participated in matching by cosine similarity or a dense layer. For interaction-based models like MatchPyramid [Pang et al., 2016a], only the matching score from the last layer's interaction pattern was extracted as a global matching score.

However we observe that there exist many types of queries: both lexical queries and conceptual queries and queries in between. For lexical queries, the exact match of query and document terms is more important, therefore the matching model is supposed to pay more attention to the term layer. However for conceptual queries a match at higher layers is more appropriate, since the representations/interactions learned at higher layers aggregate the basic terms to generalize into more abstract concepts. There are also queries in between. Therefore we argue that the use of representations/interactions of multiples layers is beneficial for a neural IR model to be flexible in processing all types of queries.

## Contributions:

The main contribution of our papers lies in a new model capable of coping with different types of queries by matching them with documents at different levels of abstraction. This

idea can be easily adopted in other deep neural models, whether they are based on representations or interactions, use CNN or RNN. Our experiments on ClueWeb09B confirm that our approach can result in superior retrieval effectiveness.

Our first paper focus on employing multiple layers in interaction-based model. In this work we proposed to use 2D convolutions to learn multiple layers of interaction patterns and a gating mechanism to dynamically combine the matching signals from those matching layers to produce a matching score. In the second paper, we extend the idea of multi-level matching to representation-based models and build representations layers by 1D convolutions. We also compare the performance of representation- and interaction-based models on comparable settings and the same evaluation collection (ClueWeb09B) to study which paradigm works better.

## 4.1. Introduction

Deep learning techniques have been successfully used first in image [Krizhevsky et al., 2012] and speech processing [Graves et al., 2013]. The key idea behind them is to learn representations to represent the content and features of images and speech. The main architecture is convolutional neural network (CNN), which aggregates representation cells at a lower-level to form a higher-level representation. It has been observed that the representations can successfully capture features from lines, forms, face components, to specific face classes (persons) at different levels in an CNN trained for face recognition. The CNN architecture demonstrates high capability of creating more and more complex and abstract features when we move up in the convolution layers.

These techniques have then been extended to text processing. To cope with the specificities of texts, deep learning techniques have been much extended, namely to incorporate the sequential dependencies between words in texts. In particular, recurrent neural networks (RNN) [Cho et al., 2014, Hochreiter and Schmidhuber, 1997] are widely used for different tasks in text processing: machine translation, question-answering, and so on.

The great success of deep learning has triggered a tremendous interest in the IR community. We saw a large number of research papers on neural IR models in the recent years [Huang et al., 2013, Shen et al., 2014, Severyn and Moschitti, 2015]. These models are based on CNN or RNN. It has been shown that RNN can be successfully used in tasks that deal with short texts such as in question-answering (i.e. to re-rank short answers) [Yang et al., 2016]. However, RNN has not been often used for long texts, in which RNN has difficulty to capture the essential part of a long text [Hochreiter et al., 2001, Pascanu et al., 2013]. For the core ad hoc IR task, CNN is the main architecture used in most of the previous studies on neural IR models.

The previous studies have proposed two main families of models: representation-based models and interaction-based models. Representation-based models such as DSSM [Huang et al., 2013], CDSSM [Shen et al., 2014], ARC-I [Hu et al., 2014] focus on learning meaningful representations through several hidden layers and apply a similarity function on the last level query and document representations to estimate relevance.

Instead of learning semantic representation of query and documents, interaction-based models calculate local interactions of each query and document term at input and learn the term-level interaction patterns through several hidden layers.

In previous work, the two approaches have been tested under different experimental conditions, making it difficult to compare them fairly. The first goal of this work is to make a fair comparison of the main families of models proposed for ad hoc search. The first question we examine is which of the representation-based and interaction-based models, when implemented in a similar manner, better suits ad hoc IR tasks.

Apart from the above issue, we also observe that previous neural IR models only employ the representations or interaction scores of the last level to produce a global interaction score. For instance, in DSSM, CDSSM and ARC-I, the query and document representations are learned separately through a series of dense or 1D convolution layers. Finally, only the query and document representations of the last layer are employed to perform a match. Similarly in interaction based model such as MatchPyramid [Pang et al., 2016a] and ARC-II [Hu et al., 2014], only the last layer of learned interaction pattern is employed to produce a matching score.

However, user's queries may be of different nature. For example, the query "fact on Uranus" (a ClueWeb query) is a lexical query for which a low-level exact match for the word "Uranus" is required. Similarly, for the query "Ron Howard' (another ClueWeb query)', an exact match at term level is also crucial since the query asks for the information about the exact person Ron Howard, not other people with a different name. For those lexical queries, an exact match on term level is crucial whereas a semantic match may run the risk of matching with other entities, leading to query drift.

On the other hand, the query "last supper painting" is a conceptual query which needs some generalized representations/interactions on higher level of the model, since this query may ask for documents containing the description of the painting or the significance of this painting in Catholicism. Similarly, the query "rice" may ask either some recipes of rice or the nutrition values of rice.

From the above examples, we can observe that user's queries require different levels of matching, from lexical to semantic levels. This is the general case for IR: users may submit queries to locate documents containing the same words, or the same or similar concepts, and there are also queries lying in between.

Since neural IR models could learn multiple levels of representations or interactions, a natural question is raised: Can different levels of representation or interaction be leveraged to cope with the needs of different queries? If yes, how to integrate them?

In order to answer this question, we extend the existing representation- and interaction-based models to multi-level matching and run extensive experiments under the same test condition to compare these models. We will first test if different layers of representation are indeed more suitable for different types of queries, and if their combination can better deal with various types of query. Our contributions are two-fold: (1) We compare the performance of representation- and interaction-based models under the same test conditions (2) We propose Multi-level Abstraction Convolution Model to cope with different types of queries by dynamically aggregating the matching signals from all layers instead of using just the last layer of output from the neural model. Our results will clearly show that interaction-based models are usually preferred to representation-based models for ad hoc search; and multi-level matching is preferred to single-level matching.

## 4.2. Related Work

### 4.2.1. Representation- and Interaction-based Models

In deep IR models, given the query $q$ and document $d$, the matching is often achieved by estimating a relevance score $rel$ of $q$ and $d$. Depending on how to produce the relevance score $S(q, d)$, previous deep IR models could be roughly divided into 2 categories: representation-based models and interaction-based models [Guo et al., 2016].

Representation-based models focus on learning meaningful semantic representations through several hidden layers and estimate a global relevance score by a matching function applied on the last level representations of the query and document. The process could be summarized in Equation 4.2.1.

$$rel(q, d) = S(\phi(q), \phi(d)) \tag{4.2.1}$$

where $\phi$ is a complex feature function to map query or document text into meaningful semantic representations through several hidden layers. S is a matching function, such as cosine or dot similarity. For example, in DSSM [Huang et al., 2013], the feature function $\phi$ is a feed forward neural network and $S$ is a cosine similarity. In CDSSM [Shen et al., 2014], $\phi$ is a convolutional network and $S$ is the cosine similarity.

Different from representation-based models, interaction-based models focus on learning salient interaction patterns from the input local interactions through a series of hidden layers. The process could be summarized in Equation 4.2.2.

$$rel(q, d) = S_n \circ S_{n-1} \circ ... \circ S_0(w(q), w(d)) \tag{4.2.2}$$

where $w$ is often a simple embedding lookup function which will extract word embeddings of corresponding terms, and the matching function is a composition of a series hidden neural transformations $S_n \circ S_{n-1} \circ ... \circ S_0$.

For example, in MatchPyramid [Pang et al., 2016a] and ARC-II [Hu et al., 2014], the feature function $w$ maps each term of query and document into word embedding vector, and the matching function $S_n \circ S_{n-1} \circ ... \circ S_0$ is a deep convolutional network of several layers to learn the matching patterns from the input interaction matrix.

Similarly, the DRMM model [Guo et al., 2016] calculates the interactions between each query term with each document term by a similarity function, and the histogram of interactions between query term $t_i^q$ and all document terms are produced. Afterwards, $n$ weight-sharing feed-forward neural networks take the histograms as input and predict $n$ matching scores, where $n$ is the length of the query. Finally the $n$ scores are aggregated through an aggregating gate to produce a global matching score.

The two families of approaches have been extensively tested. [Huang et al., 2013] and [Shen et al., 2014] showed that representation-based models trained on clickthrough data could successfully produce superior effectiveness than a traditional model (BM25) on document title retrieval. However, [Pang et al., 2016a] and [Guo et al., 2016] showed that DSSM and CDSSM were far less effective than BM25 on ad hoc retrieval and interaction-based models performed better. It is difficult to draw a solid conclusion from these experiments because the retrieval tasks and test conditions are very different. This is the very motivation of our paper - to compare the two approaches under the same test condition. In this study, in order to understand the contribution of representation-based and interaction-based models, we test them separately in this paper.

The effects of representation-based and interaction-based models are believed to be complementary: they can respectively achieve a high-level global matching or a fine-grained low-level matching. Therefore, they are combined in some models. For example, the DuetNet [Mitra et al., 2017] incorporates the strengths of both representation-based and interaction-based models by explicitly building 2 sub-models. In this paper, however, we intend to compare the two learning and matching schemas directly, without mixing up other aspects. So, we do not consider such a combined model in this study.

In general, a neural IR model tends to create a high-level representation or matching pattern along the convolution layers. It has been noticed that such a model may fail to deal with lexical queries. To address this issue, the AttR-Duet model [Xiong et al., 2017a] exploits both word and entity matching features like BM25 and TF-IDF scores and builds two 1D CNN models to produce 2 matching scores, and the word and entity matching scores are linearly combined with attention weights learned by a separate attention model. [Xiong et al., 2017a] shows the importance of low-level lexical features. In our study, we will also

combine lexical and semantic matchings, but in a different architecture. We propose to combine the matching scores at multiple levels of convolution.

From the above analysis, we can observe that previous neural IR models only employed the final level or low-level representation/interaction score to estimate a global relevance score. However, as discussed in previous section, user's queries may be of different nature, which may require matching at different levels of abstraction. Therefore we propose to investigate the possibility of integrating multi-level matching into representation-based and interaction-based models. The details will be discussed in the following sections.

### 4.2.2. Weak Supervision

The training of deep models requires a lot of labeled data. However in reality there's often a lack of such labeled (query, document, relevance) triples. To address this problem, the weak supervision framework was proposed in [Dehghani et al., 2017]. The idea of weak supervision is to use a simple, unsupervised retrieval model such as BM25 to estimate relevance scores for query-document pairs, and use the computed relevance score as a weak label.

The advantage of weak supervision is that it could generate large amount of weakly supervised training data at low cost. Once we have access to a certain amount of queries (e.g. query log), we can launch those queries into a retrieval model such as BM25 against a document collection to generate large amount of (query, document, score) triples. Even though the generated labels for the query-document pairs are not perfect, we hope that the large amount of the weak supervision training examples could compensate the imperfections of the weak labels.

In this study, we employ the AOL query log presented in Section 3.2, and apply the weak supervision framework [Dehghani et al., 2017] to generate our training data.

## 4.3. Representation-based Multi-level Abstraction Convolution Model

In this section, we will present the representation-based Multi-level Abstraction Convolution Model. In order to compare the performance of combining multiple levels of abstractions with the tradition approach of using only the last layer of query and document representations, we first build representation-based single-level matching model. Afterwards, we will introduce our proposed Multi-level Abstraction Convolution Model (MACM).

### 4.3.1. Representation-based Single-level Convolution Model

The architecture of the Representation-based Single-level Convolution Model is presented in Fig 4.1. This model is similar to CDSSM [Shen et al., 2014] without the word transfor-

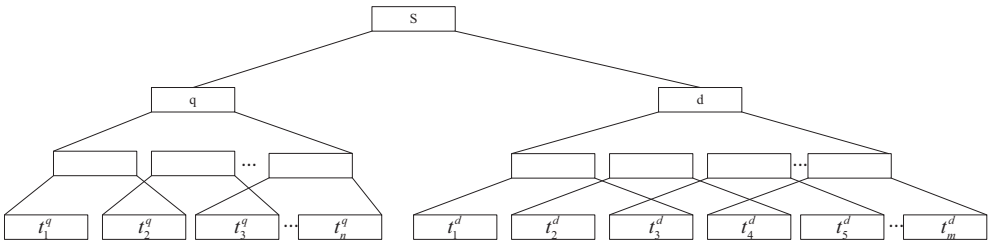



**Fig. 4.1.** Representation-based Convolutional Model

mation to letter-trigram. In this model, the query terms and document terms are combined into more abstract representations through 1D convolutions [Zhou et al., 2015].

Let $q$ and $d$ denote the query and document respectively. In this model, $q$ and $d$ are represented by a sequence of word embeddings $q = [t_1^q, t_2^q, ..., t_n^q]$ and $d = [t_1^d, t_2^d, ..., t_m^d]$, where $t_i^q$ and $t_j^d$ represent the word embedding of the $i^{th}$ query term and the $j^{th}$ document term respectively. Then, a series of 1D convolutions are performed to combine the word representations into more abstract representations as follows.

$$C_i^{q,(1)} = f(W_1^q * [t_{i-w(1)}^q; ..; t_{i+w(1)}^q] + b_1^q) \tag{4.3.1}$$

$$C_i^{q,(k)} = f(W_k^q * [C_{i-w(k)}^{q,(k-1)}; ...; C_{i+w(k)}^{q,(k-1)}] + b_k^q),\ k = 2,...,L \tag{4.3.2}$$

$$C_i^{d,(1)} = f(W_1^d * [t_{i-w(1)}^d; ..; t_{i+w(1)}^d] + b_1^d) \tag{4.3.3}$$

$$C_i^{d,(k)} = f(W_k^d * [C_{i-w(k)}^{d,(k-1)}; ...; C_{i+w(k)}^{d,(k-1)}] + b_k^d),\ k = 2,...,L \tag{4.3.4}$$

where $W_k^q$, $b_k^q$, $W_k^d$, $b_k^d$ are the weight and bias for the query and document of the $k^{th}$ layer respectively; $t$ represents the input word embedding layer and $C_i^{q,(k)}$, $C_i^{d,(k)}$ are the $i^{th}$ convolved vectors of the $k^{th}$ layer; $2w(k)+1$ is the window size for the $k^{th}$ layer; $f$ is a non-linear transformation. Once the final level representations of the query $C^{q,(L)}$ and the document $C^{d,(L)}$ are obtained, a cosine similarity function is applied to estimate the global matching score $S_L$ as follows.

$$S_L = Cos(C^{q,(L)}, C^{d,(L)}) \tag{4.3.5}$$

It is worth noting that similar to the existing representation-based models, in this Representation-based Single-level Convolution Model, only the last layer of query and document representations are employed to perform a match. However as discussed in Section 4.1, user's query may be of different types, ranging from lexical queries to conceptual queries and those in between. Therefore it is more suitable to employ multiple levels of representations to perform matching in representation-based model.

### 4.3.2. Representation-based Multi-level Matching Model

The Representation-based Multi-level Matching Model (MACM-Rep) is an extension from the representation-based model described in the previous section, in which we will define a matching function at every level of representation. The architecture is shown in 4.2. In this architecture, we first learn multiple layers of query and document representations

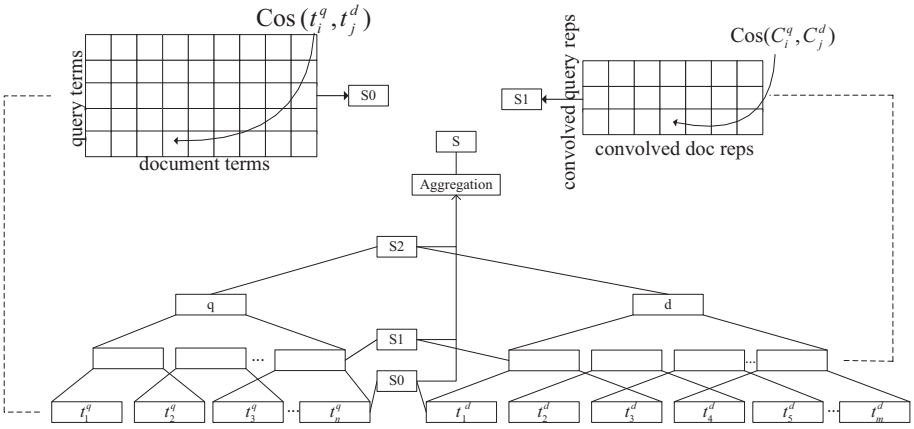



**Fig. 4.2.** Representation-based Multi-level Matching Model

through 1D convolutions as the case in the Single-level Matching Model presented in previous section. Afterwards, instead of computing the only one matching score ($S_2$) at the last level between query and document, we also define matching scores $S_0$, $S_1$ at the input term level and the intermediate level. The term level matching score $S_0$ is expected to capture some signals of term matching. It is possible to replace this function by a traditional matching function such as BM25, as in [Severyn and Moschitti, 2015], but, as we explained, we want to keep the whole architecture within the neural framework in this study. The intermediate level matching score $S_1$ is expected to capture some concept learned by generalizing term representations.

Note that the architecture we use only contains three levels. It would be possible to add more levels into it. However, more levels would mean more parameters to train and more training data. In our case, as the amount of training data is limited, we limit our investigation to an architecture with three levels.

The matching scores at different levels are determined as follows: For the $i^{th}$ level, we construct a interaction matrix between the query and document representation vectors.

$$I_{ij}^{(0)} = Cos(t_i^q, t_j^d) \tag{4.3.6}$$

$$I_{ij}^{(k)} = Cos(C_i^{q,(k)}, C_j^{d,(k)}),\ k = 1,..,L-1 \tag{4.3.7}$$

where $I^k$ is the interaction matrix of the $k^{th}$ level. $t_i^q$ and $t_j^d$ are the word embeddings of the $i^{th}$ query term and the $j^{th}$ document term. $C_i^{q,k}$ and $C_j^{d,k}$ are the $i^{th}$ convolved vector for the query at and the $j^{th}$ convolved vector for the document at layer layer $k$ respectively.

**Level-importance Score:** Once the interaction matrices are produced, a series of level-importance scores are extracted as follows: We take the top $P$ interactions across each row $u$ of $I^{(k)}$ ($P$ is set to 5 in this study) and average them to obtain a scalar value $M_u^{(k)}$. This $M_u^{(k)}$ represents the strongest interactions between the $u^{th}$ query representation and the document. The idea behind is that the matching score only depends on a few matching spots in the document instead of the entire document. We then sum up $M_u^{(k)}$ for every query term/representation to obtain a matching score $S^{(k)}$ for the level $k$. The final level matching score $S^{(L)}$ is estimated by the cosine similarity of the global query and document representation. The process could be summarized as follows.

$$M_u^{(k)} = \frac{1}{P} \sum_{p=1}^{P} \operatorname*{top\,P}_{v=1..m} I_{uv,p}^{(k)} \tag{4.3.8}$$

$$S^{(k)} = \sum_{u=1}^{n} M_u^{(k)} \tag{4.3.9}$$

The intuition of designing the level-importance scores is to quantify the importance of each level. If the score of one specific layer is higher, we would pay more attention to the matching score of this layer when aggregating the matching scores of different levels.

**Aggregating Gate:** Once the scores of each level are extracted, they are aggregated through a softmax gate to produce the global matching score $S$ as follows.

$$\beta_k = \frac{exp(\alpha_k S^{(k)})}{exp(\sum_{k=0}^{L} \alpha_k S^{(k)})} \tag{4.3.10}$$

$$S = \sum_{k=0}^{L} \beta_k S^{(k)} \tag{4.3.11}$$

where $\alpha_i$ are learnable parameters, $S^{(k)}$ is the matching score of the level $k$.

We have also tested other aggregation strategies such as direct concatenation of the each level's matching score and pass it into the last output MLP, or a weighted average of each level's matching score (without the guidance of level-importance score). They don't work as well as the gated aggregation described above. The advantage of the proposed gated aggregation is that the provided level-importance scores serve as guidance to the aggregation process: the higher the layer's interaction is, the more attention we will pay to this layer.

### 4.3.3. Training

For both single-level and multi-level matching models, we employ a pairwise training scheme. The loss is defined in Equation 4.3.12, where $S(Q,D_+)$ and $S(Q,D_-)$ are the predicted scores for positive and negative example, $\Theta$ includes all trainable parameters of the model.

$$L(Q,D_+,D_-;\Theta) = max(0, 1 - (S(Q, D_+) - S(Q,D_-))) \tag{4.3.12}$$

## 4.4. Interaction-based Multi-level Abstraction Convolution Model

Same as for the representation-based model, to compare the effectiveness of employing multiple levels of interactions between query and document over using only the last level of query-document interaction, we first build the Interaction-based Single-level Convolution Model. Afterwards, we will introduce our proposed Interaction-based Multi-level Convolution Model.

### 4.4.1. Interaction-based Single-level Convolution Model

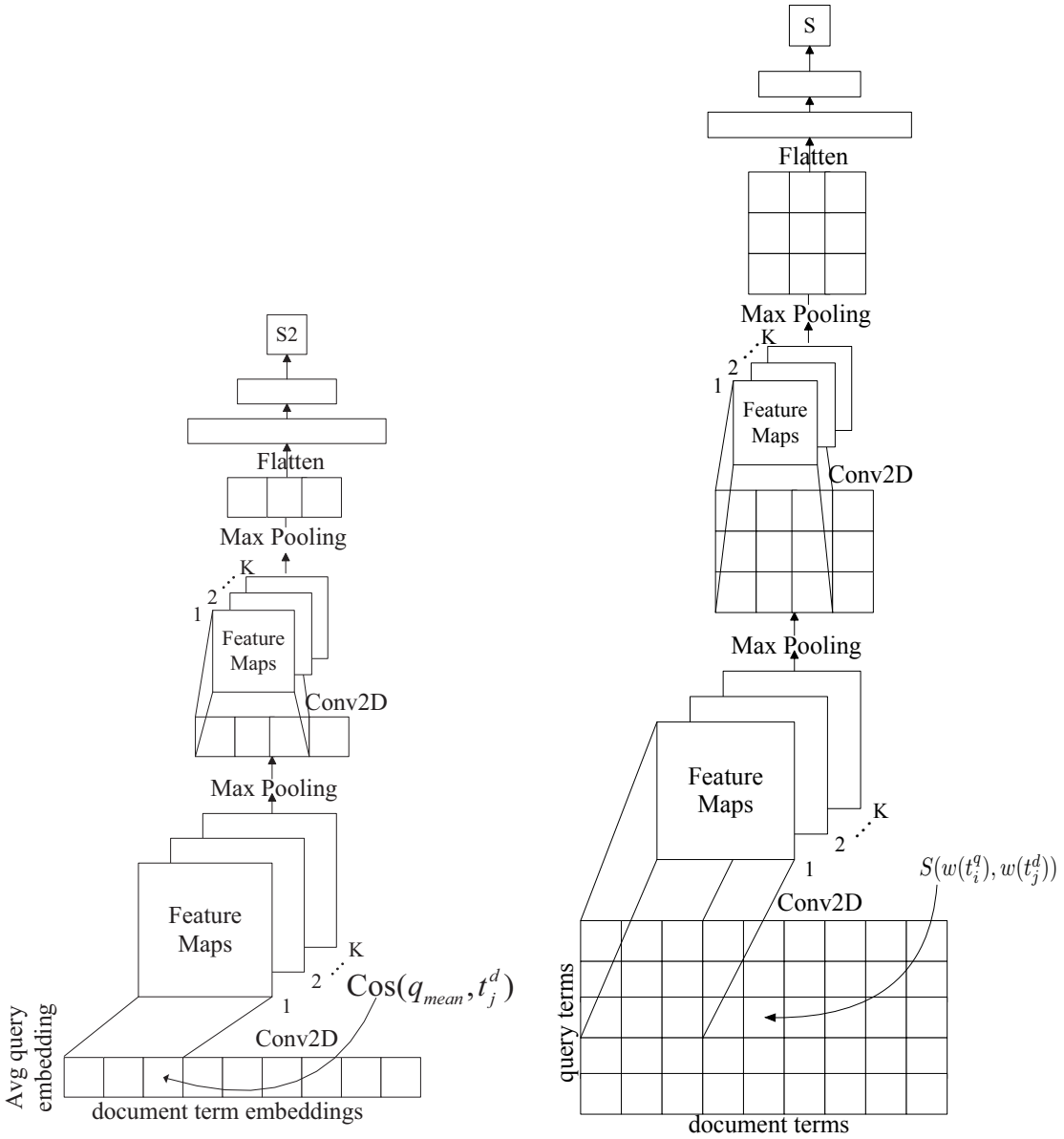
The interaction-based model we implement is inspired by MatchPyramid, which has several convolution and pooling layers on top of the basic interaction matrix between document and query terms [Pang et al., 2016a]. This architecture represents the essence of the family of interaction-based models (although there are some quite important details in other alternative models). The architecture of the Interaction-based Convolutional Model is presented in Fig 4.3.

At input, an interaction grid $I$ is constructed, which can be done in different ways (explained below). Afterwards, several convolutional layers and max-pooling layers [Pang et al., 2016a] are constructed to learn the underlying interaction patterns. Finally, a MLP is added on top of the last max-pooling layer to extract a relevance score as global matching score. There are 2 ways to build the interaction grid $I$: Either we build the interaction between the global query representation and each document term representation (1D interaction) or we build the interaction between each query term and document term (2D interaction). In this study, both 1D and 2D interactions are explored.

In 1D interaction-based model, the interaction $I$ reduces to a vector. We first calculate a global query representation by taking the average of all query term embeddings as follows.

$$q_{mean} = \frac{1}{n}\sum_{i=1}^{n} t_i^q \tag{4.4.1}$$

where $n$ is the query length, and $t_i^q$ is the embedding vector of the $i^{th}$ query term.




**(a)** 1D Inter **(b)** 2D Inter

**Fig. 4.3.** Intercation-based Convolutional Model

Then the interaction vector $I$ is constructed by calculating the cosine similarity between $q_{mean}$ and each document term embedding $t_j^d$ as follows.

$$I_j = cos(q_{mean}, t_j^d) \tag{4.4.2}$$

In 2D interaction-based model, the interaction $I$ is a matrix with each entry $I_{ij}$ being the cosine similarity of query term $t_i^q$ and document $t_j^d$ calculated as follows.

$$I_{ij} = Cos(t_i^q, t_j^d) \tag{4.4.3}$$

Once the interaction grid $I$ is constructed, a series of convolutions and max-poolings are performed as follows.

$$C_1^k = f(W_1^k * I + b_1^k),\ k = 1,..,K \tag{4.4.4}$$

$$P_1^k = max\_pool(C_1^k),\ k = 1,..,K \tag{4.4.5}$$

$$C_i^k = f(W_i^k * P_{i-1} + b_i^k),\ i = 2,..,L,\ k = 1,..,K \tag{4.4.6}$$

$$P_i^k = max\_pool(C_i^k),\ i = 2,..,L,\ k = 1,..,K \tag{4.4.7}$$

where $C_i^k$ is the feature map $k$ of the $i^{th}$ convolved layer; $I$ is the input interaction matrix; $W_i^k$ and $b_i^k$ are the kernel and bias of layer $i$ for the feature map $k$; $L$ is the number of convolution layers, and $K$ is the number of feature maps; $f$ is a non-linear mapping; and $*$ represents the convolution operator.

In order to determine the global matching score, the last max-pooled layer is flattened into a 1D vector and fed into a fully connected MLP to output a scalar score $S$

$$t = g(W^h P_L + b^h) \tag{4.4.8}$$

$$S = h(W^s h + b^s) \tag{4.4.9}$$

where $t$ and $S$ represent the hidden layer of the MLP and the matching score respectively; $W^h$, $b^h$, $W^s$, $b^s$ are the weights and biases for the hidden and scoring layer; $g$ and $h$ are non-linear mappings.

### 4.4.2. Interaction-based Multi-level Matching Model

Different from representation-based models, instead of learning representations for query and document, interaction-based models focus on learning the underlying matching patterns from the input interaction signals through several hidden layers [Pang et al., 2017, Hu et al., 2014]. In order to investigate the effectiveness of multi-level matching in interaction-based model, we also build an Interaction-based Mulit-level Matching Model (MACM-Inter). The model is presented in Fig 4.4.

The convolution-pooling part (left part) of the model is identical to that described in the previous section. What is added is a series of matching scores at every level of convolution, as well as an aggregation layer to combine these scores into a global score. The details are presented as follows:

**Level-importance Score:** For the input interaction matrix $I$ and each convolved layer $C_i$, a scalar feature $M^{(i)}$ will be calculated. For the input interaction matrix $I$, we take the max interaction values $M_u^{(0)}$ across each row $u$, which represents the max matching intensity across all document terms for the query term $t_u^{(q)}$. Afterwards, we sum up all the $M_u^{(0)}$s for each query term $t_u^{(q)}$ and get the global maximum interaction value $M^{(0)}$ for the whole query with respect to the document. This quantity reflects the word-level matching between

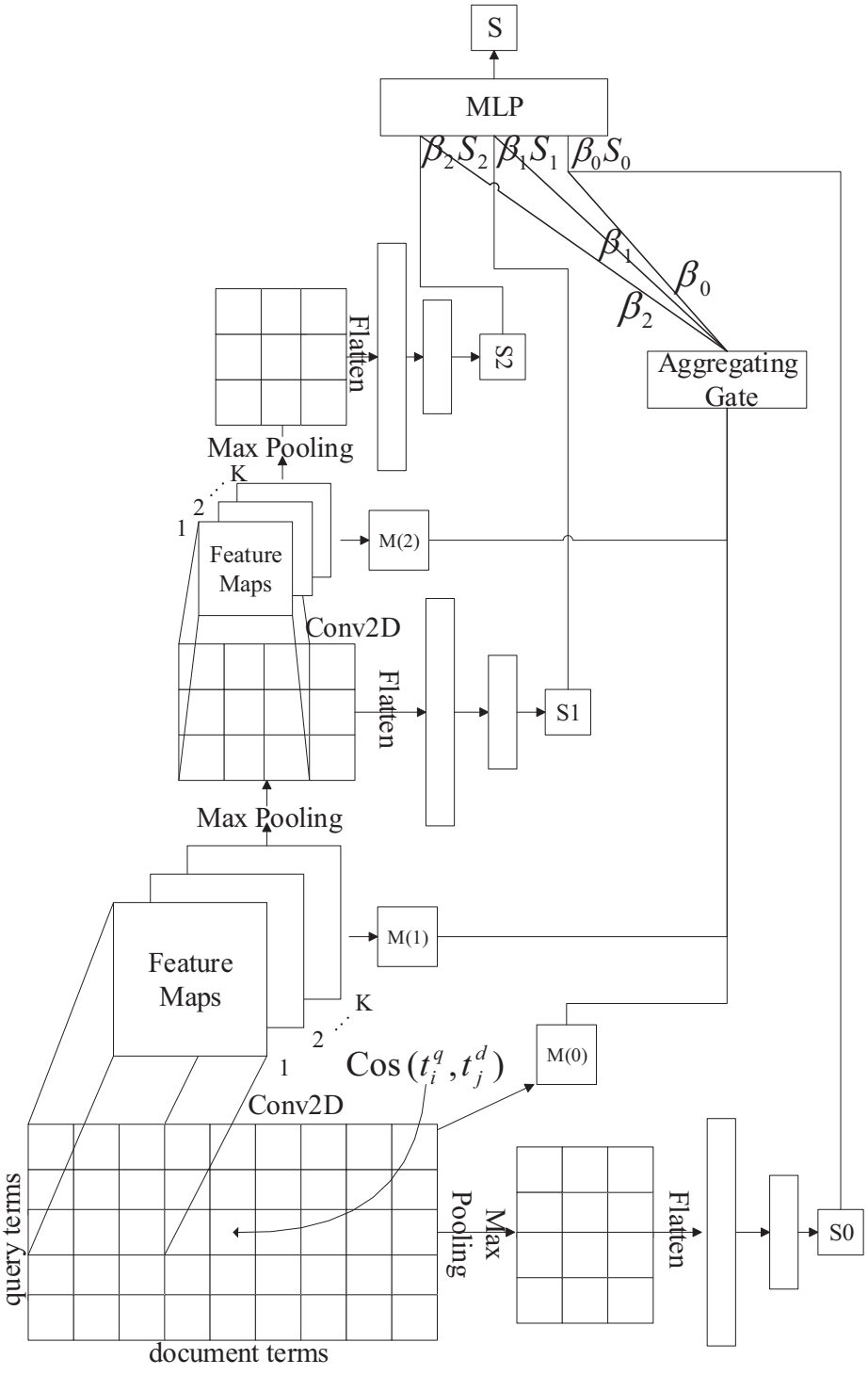



**Fig. 4.4.** Interaction-based Multi-level Matching Model

document and query. The process could be summarized as follows.

$$M_u^{(0)} = \max_{v=1..m} I_{uv},\ u = 1,..,n \tag{4.4.10}$$

$$M^{(0)} = \sum_{u=1}^{n} M_u^{(0)} \tag{4.4.11}$$

For each feature map $C_i^{(k)}$ in the convolved layer $i$, we proceed in the same way to obtain a $M_{(k)}^{(i)}$ for this specific feature map $k$. Then we average the $M_{(k)}^{(i)}$s to obtain the global $M^{(i)}$ for this convolution layer. The process could be summarized as follows:

$$M_{u,(k)}^{i} = \max_{v=1..m} [C_i^{(k)}]_{uv},\ u = 1,..,n,\ k = 1,..,K \tag{4.4.12}$$

$$M_{(k)}^{(i)} = \sum_{u=1}^{n} M_{u,(k)}^{(i)},\ k = 1,..,K \tag{4.4.13}$$

$$M^{(i)} = \frac{1}{K} \sum_{k=1}^{K} M_{(k)}^{(i)} \tag{4.4.14}$$

The motivation of designing the $M$ features is to try to capture the importance of each interaction level. Intuitively, if the maximum interaction intensity at the level $i$ is high, we would rely more the interaction score of this level. Therefore the $M$ features provide evidence of importance when we assign gating weights to combine the interaction scores of every abstraction level.

**Aggregating Gates:** After obtaining the level-importance score $M$ as described above, those values are normalized through a softmax gate as follows.

$$\beta_i = \frac{exp(\alpha_i M^{(i)})}{exp(\sum_{j=0}^{L} \alpha_j M^{(j)})} \tag{4.4.15}$$

where $\alpha_i$ are learnable parameters, $M^{(i)}$ are the $M$ values for each convolution layer $i$, and $L$ is the total number of convolution layers.

The interaction scores $S_i$ are then weighted and concatenated as $[\beta_0 S_0,..,\beta_L S_L]$ to be fed into a MLP aggregator to obtain an overall relevance score:

$$S = f(W[\beta_0 S_0,..,\beta_L S_L] + b) \tag{4.4.16}$$

### 4.4.3. Training

For both single-level and multi-level matching models, we employ a pairwise training scheme. The loss is defined in Equation 4.4.17, where $S(Q,D_+)$ and $S(Q,D_-)$ are the predicted scores for positive and negative example, $\Theta$ includes all trainable parameters of the model.

$$L(Q,D_+,D_-;\Theta) = max(0, 1 - (S(Q, D_+) - S(Q,D_-))) \tag{4.4.17}$$

## 4.5. Experimental Setup and Datasets

In this section, we will present the collection used in this study, evaluation metrics and experimental settings. Those experiments aims to investigate the following research questions: (1). In a comparable setting, does interaction-based model generally outperform representation-based model or the reverse is true? (2). Is multi-level matching beneficial to

ad-hoc retrieval tasks compared with single-level matching models? If so, how could we combine the matching signals from multiple levels? We will present the collection and evaluation metrics used in this study and the experimental setups in the following subsections.

### 4.5.1. Collection

We employed the ClueWeb09B collection as our document collection in this study. The details of this collection has been described in Section 3.1. As mentioned in Section 3.2, the training of a deep IR model requires a lot of training data , and very often they are not easily accessible at a large scale. Therefore, we follow the Weak Supervision [Dehghani et al., 2017] framework to generate the training triples used in this study. We employ the queries 1-50 in this collection as validation queries and 51-200 as test queries.

To generate the training triples, we employ the AOL Query Log described in Section 3.2 as our training query and BM25 model implemented in Indri[1] as our weak supervisor. For each query we retrieve the top 50 documents using BM25 model with default parameters ($k_1 = 1.2$,$b = 0.75$,$k_3 = 1000$). Afterwards, we employ a pair-wise training scheme and convert the retrieved documents to positive and negative training documents as follows: for a given query, we randomly sample 2 documents and regard the one with higher BM25 score as positive document, the other one as negative document.

### 4.5.2. Evaluation Metrics

In this study, we employ Mean Average Precision (MAP), Normalized Discounted Cumulative Gain (NDCGs) as evaluation metrics. They are presented in detail in Section 3.3. In particular we calculate NDCG at cut-off position 1, 3, 10, 20. The MAP is a binary metric and aims to evaluate the general ad-hoc performance. NDCGs at different cut-off positions simulate the real search scenario where a user would only want to browse the top-k ranked documents.

Finally for statistical tests, we perform paired t-test ($p < 0.05$) of each of our proposed model against the BM25 baseline.

### 4.5.3. Experimental Setup

In this subsection, we will present the detailed experimental setup of our models.

**General Settings:** We employ the GloVe.6B.300d pre-trained embeddings presented in Section 3.2 to initialize the embedding layer of all our models. The dimension of each embedding vector is 300. Since the interaction-based models mainly focus on learning the interaction patterns whereas the representation-based models relies on learning good representations,

[1] https://www.lemurproject.org/indri/

we fix the embeddings for interaction-based models and continue to fine-tune them during training for representation-based models.

We set the max query length and document length to be $n = 15$, $m = 1000$ and apply zero paddings as it is done in [Pang et al., 2016a]. The maximum query length limit is enough to cover most of the queries in the training set and all the queries in the validation and test sets (4 and 5 words respectively). For documents, the parts that exceed the limit are cut off. Since document in this collection are very long, we omit the out-of-vocabulary (OOV) terms in document.

As for evaluation, we employ reranking scheme during validation and test: For each validation or test query, we first retrieve top 1000 documents by BM25 and then rerank them by our proposed model.

**Representation-based Model:** For experiments on the representation-based models, we limit the max number of convolution layers $L$ to 3 and fix the hidden size to be 128. For 2-convolutional-layered model, the convolutional window sizes are set to $[3, 13]$ for query side (i.e. 3 for the first layer and 13 for the second layer), $[3, 998]$ for document side, and all strides are set to 1. For 3-convolutional-layered model, the convolutional window sizes are set to [3,5,9] and strides are set to [1,1,1] for query side, and [3,10,198] and strides are set to [1,5,1] for document side.

**Interaction-based Model:** For experiments on the 1D interaction-based model, in order to determine best the convolution filter size, we conducted preliminary tests with different filter size combinations on the validation set. For two-layered 1D interaction-based model, we tested it with [2,4], [3,5], [4,6] and for three-layered model, we tested it with [2,4,7], [3,5,9], [4,6,10]. The optimal filter sizes are set to $[3, 5]$ for the two layered model, and the optimal filter sizes are set to [3,5,9] for the three-layered model. The pooling size is set to 2 for each max-pooling layer of both models. The number of feature maps is set to 128 for both models.

For experiments on the 2D interaction-based model, we limit the max number of convolution layers $L$ to be 2 due to memory limit and fix the number of feature maps to $[32, 16]$ for the 2 convolution layers. We fix the pooling size of all max pooling layers to be (2,2). We also conducted preliminary tests to determine the optimal filter shapes: we tested 2D convolutional filters of shape [(1,3),(2,5)],[(2,2),(4,4)],[(3,3),(5,5)],[(3,3),(5,10)] for the two convolution layers. The optimal filter shapes for the 2 convolution layers are set to (3,3) and (5,5) according to this preliminary test.

## 4.6. Experimental Results

In this section, we will present the experimental results of our proposed models.

### 4.6.1. Single-level Representation- vs Interaction-based Models

First of all, one of our research question is to investigate whether an interaction-based model generally outperforms a representation-based model when trained and evaluated on the same training and testing set.

We first present the experimental results of Single-level representation- and interaction-based model presented in Section 4.3.1 and 4.4.1. They follow the idea of existing models in literature which only employ the last layer query and document representations or query-document interactions to produce a matching score.

The experimental results on the test set are presented in Table 4.1, where Rep-2L, Rep-3L represent the single-level representation-based models with 2 and 3 convolution layers respectively. Inter-1D-2L, Inter-1D-3L and Inter-2D-2L represent single-level 1D interaction models with 2 and 3 convolution layers and single-level 2D interaction model with 2 convolution layers respectively.

During training, for each model, we monitor its performance of MAP on the held-out validation set and only select the best model based on validation performance. This selected best model is then evaluated on the test set.

**Table 4.1.** Results of Representation-based and Interaction-based Models on Test Set[1]

| Model | MAP | NDCG@1 | NDCG@3 | NDCG@10 | NDCG@20 |
|---|---|---|---|---|---|
| BM25 | 0.0879 | 0.1178 | 0.1359 | 0.1356 | 0.1394 |
| Rep-2L | 0.0121 | 0.0019* | 0.0024 | 0.0042 | 0.0053 |
| Rep-3L | 0.0145* | 0.0158* | 0.0143* | 0.0143* | 0.0149* |
| Inter-1D-2L | 0.0663* | 0.0927 | 0.0846* | 0.0912* | 0.0967* |
| Inter-1D-3L | 0.0632* | 0.0662* | 0.0922* | 0.0904* | 0.0957* |
| Inter-2D-2L | **0.0884** | **0.1485*** | **0.1423** | **0.1411** | 0.1389 |

From Table 4.1, we observe that representation-based models don't perform well. One possible reason is that it is difficult to learn good global representation of the document which is often very long. Since the model employs only the final level representations to estimate relevance, the performance will be influenced by the quality of the global representations. Intuitively, it is very difficult, even impossible, to represent every aspect of a long document in a single dense vector, and that the vector should be appropriate to match with any related query. This may be too much to ask. The poor effectiveness of representation-based models has been observed in several previous studies [Guo et al., 2016, Pang et al., 2016a]. Our observation is consistent.

[1]* means statistically significant difference with BM25

We also observe that the Inter-2D-2L model could get competitive result with the BM25 model with the help of weak supervised data. The sharp contrast between interaction-based and representation-based models suggests that the former can better capture useful matching signals than the latter. In contrast to a global representation for a document, the interaction-based models try to determine local matching signals between document and query. Local matching signals are known to be important in IR - in fact, all the traditional models are built on similar local matching signals. This also correspond to our understanding of the general search tasks: a document is relevant often because parts of its content are relevant. The local matching signals reflect this very principle. This observation has also been made in some previous studies such as [Guo et al., 2016].

It could be possible that the difference between the Inter-2D model and the Rep models is due to the use of 1D and 2D convolutions. To better understand this aspect, we compare Inter-1D with Rep models, which all use 1D convolution. In Fig. 3, we can see that Inter-1D models (with 1 or 2 layers) clearly outperform representation-based models. This observation shows that the main difference is due to the learning object - representation or interaction pattern. However, we also observe that Inter-1D models have lower effectiveness than Inter-2D model. This shows that the use of 2D convolution on the basis of term interactions may capture better matching signals than the 1D convolution based on the interactions between the whole query and each document term. The granularity of the interactions matters.

### 4.6.2. Multi-level Representation- vs Interaction-based Models

The experimental results of multi-level matching models evaluated on the test set are presented in Table 4.2, where $Rep\_xL\_Sy$ represents the representation-based model with $x$ convolutional layers, and $Sy$ scores involving in matching. For instance, S0+S1 means aggregating S0 and S1 scores. $Inter - Sx$ is the interaction-based model with matching score $Sx$ of level $x$ participating in matching. During training, for each model, we monitor its performance of MAP on the held-out validation set and only select the best model based on validation performance. This selected best model is then evaluated on the test set.

From Table 4.2, we can observe that for both representation- and interaction-based models, the ones employing multi-level matching scores outperform the ones employing only the last level matching score. This result indicates that multi-level matching signals are important to ad hoc tasks. It is also worth noting that with multi-level matching mechanism, the interaction-based model continues to perform better than representation-based models. This result is consistent with the models with single-level matching presented in Table 4.1, and further confirms that it is preferable to employ interaction-based model in ad hoc search

**Table 4.2.** Results of Multi-level Matching Models on Test Set[1]

| Model | MAP | NDCG@1 | NDCG@3 | NDCG@10 | NDCG@20 |
|---|---|---|---|---|---|
| BM25 | 0.0879 | 0.1178 | 0.1359 | 0.1356 | 0.1394 |
| Rep_0L_S0 | 0.0614* | 0.0862 | 0.0861* | 0.0936* | 0.0953* |
| Rep_1L_S0+S1 | 0.0401* | 0.0857 | 0.0697* | 0.0690* | 0.0709* |
| Rep_2L_S2 | 0.0121 | 0.0019* | 0.0024 | 0.0042 | 0.0053 |
| Rep_2L_S0+S1+S2 | 0.0503 | 0.0899* | 0.0882 | 0.0875 | 0.0857 |
| Rep_3L_S3 | 0.0145* | 0.0158* | 0.0143* | 0.0143* | 0.0149* |
| Rep_3L_S0+S1+S2 | 0.0386* | 0.0452* | 0.0609* | 0.0658* | 0.0660* |
| Rep_3L_S0+S1+S2+S3 | 0.0686* | 0.0837* | 0.0971* | 0.1080* | 0.1092* |
| Inter-S0 | 0.0546* | 0.1218 | 0.1020* | 0.0991* | 0.0938* |
| Inter-S1 | 0.0789* | 0.1218 | 0.1254 | 0.1283 | 0.1272* |
| Inter-S2 | 0.0884 | 0.1485* | 0.1423 | 0.1411 | 0.1389 |
| Inter-S0+S1+S2 | **0.0928** | **0.1610*** | **0.1483** | **0.1431** | **0.1424** |

tasks. In fact, the interaction-based model with multi-level matching $Inter - S0 + S1 + S2$ can even outperform the BM25 baseline.

## 4.7. Discussions and Analysis

In this section, in order to better understand the advantage of multi-level matching over single-level matching, we will discuss the experiment results, give some case studies based on some typical queries in the test collection and give the complexity analysis of our proposed Multi-level Abstraction Convolution Model.

### 4.7.1. Learning Curves

**Single-level Matching Models:** To better understand whether interaction-based models generally outperform representation-based models on the same settings, we present the learning curves of single-level representation- and interaction-based models mentioned in Section 4.3 and 4.4 in Fig. 4.5.

All the curves measure the performance of MAP of different models on the validation set and were logged periodically during training. We only retain the best model based on validation performance and employ it to evaluate its test performance on test data. The dark blue line marks the MAP of BM25 baseline which doesn't need learning and therefore is a straight line. This sets a reference for us to understand how the deep IR model performs compared with BM25. The green curve represents the 2D interaction-based model with 2 convolutional layers (Inter-2D-2L). The yellow curve and light blue curve represent the

[1]* means statistically significant difference with BM25

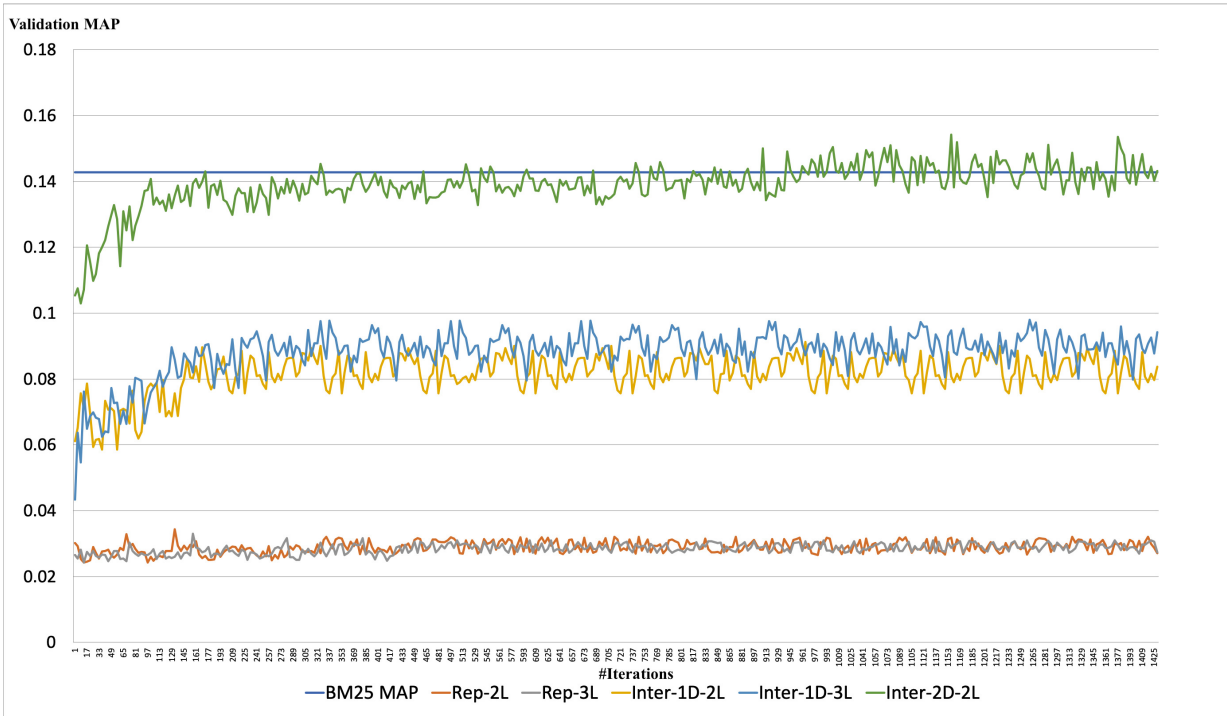



**Fig. 4.5.** Single-level Models Learning Curve on Validation Set

1D interaction-based models with 2 and 3 convolutional layers respectively (Inter-1D-2L, Inter-1D-3L). The orange and gray curve represent representation-based models with 2 and 3 convolutional layers respectively (Rep-2L, Rep-3L).

From Fig 4.5, we can observe that for the Inter-2D-2L, at the beginning of the training, the model performance is lower than the MAP of BM25 model. As training goes on, the performance would increase and approach the MAP of BM25. This observation confirms the effectiveness of weak supervision. For 1D interaction models (Inter-1D-2L and Inter-1D-3L), the performance also first increases at the start of the training and then stagnates, however, the performance at convergence is not as good as 2D interaction-based models. This indicates that using term-to-term, 2D query-document interaction matrix as input is better than compressing the query terms into one vector and match with every document term.

We also observe that for the representation-based models (Rep-2L and Rep-3L), the performance could not catch up with BM25, showing that learning representation is more difficult than learning interaction patterns.

**Multi-level Matching Models:** To compare the multi-level matching representation-based model with interaction-based models, we also plot the learning curve of multi-matching representation- and interaction-based models in Fig 4.6.

Same as Fig. 4.5, all the curves measure the performance of MAP of different models on the validation set and were logged periodically during training. The dark blue line marks

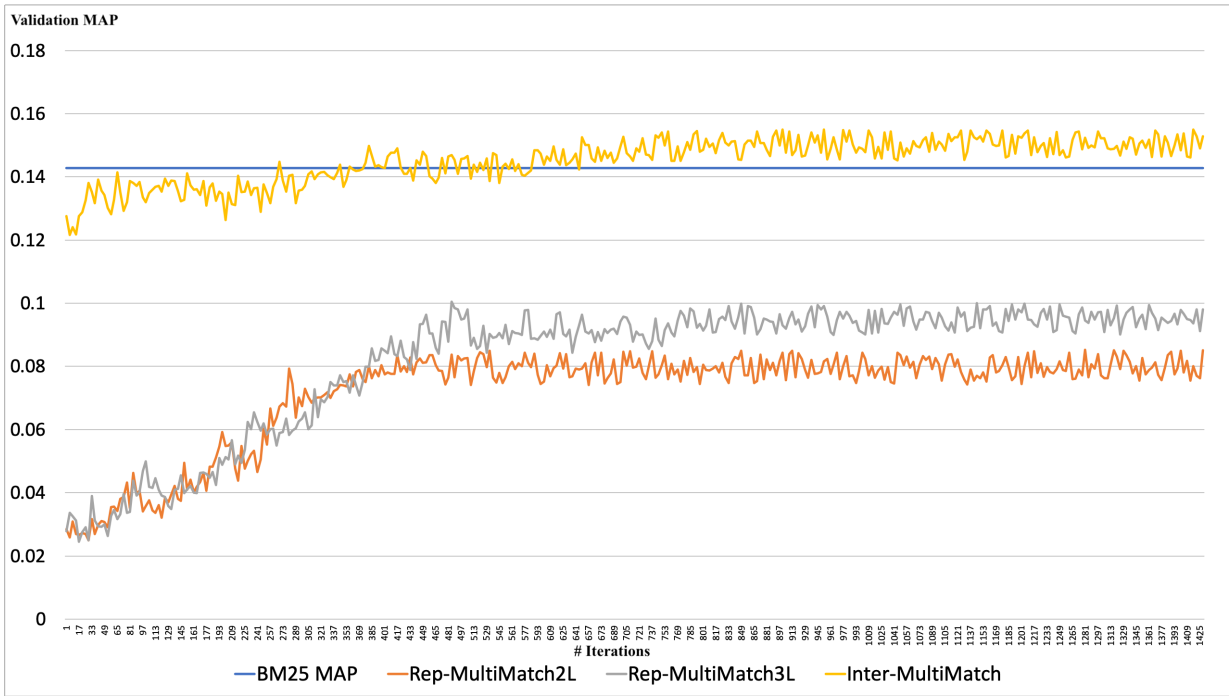



**Fig. 4.6.** Multi-level Models Learning Curve on Validation Set

the MAP of BM25 baseline. The yellow curve represents the performance of interaction-based multi-level matching model with 2 convolutional layers (Inter-S0+S1+S2). The orange and gray curve represent representation-based multi-level matching models with 2 and 3 convolutional layers respectively (Rep_2L_S0+S1+S2 and Rep_3L+S0+S1+S2+S3).

From Fig 4.6, we can observe that in the case of multi-level matching, interaction-based model still outperforms representation-based model, which indicates the difficulty of training good representations in representation-based models, even when multiple matchings are allowed. In fact, although multiple matching scores allow us to match a query and a document at different levels of abstraction, they are still based on global representations of the document. These latter continue to have difficulties to capture the important matching elements for different specific queries.

For the interaction-based multi-level matching model (Inter-S0+S1+S2) we can observe that the validation performance at the beginning of training was below that of BM25, then as training goes on, the validation performance increases and at some point surpass that of

BM25. This indicates that with the help of matching scores from multiple layers, the model could outperform a traditional model under weak supervision.

### 4.7.2. Case Studies

To better understand the usefulness of multi-level matching in representation-based models, we compare the representation-based multi-matching model with some base models on some representative queries in Table 4.3, where Rep-3L-MultiMatch is the model with matching scores of every level ($S_0 + S_1 + S_2 + S_3$).

**Table 4.3.** NDCG@10 of Representative Queries for Rep Models

| Topic_num | Query | Rep-3L_S0 | Rep-3L_S3 | Rep-3L-MultiMatch |
|---|---|---|---|---|
| 62 | Texas border patrol | 0.0496 | 0.0092 | **0.0585** |
| 73 | Neil Young | 0.0600 | 0.0221 | **0.0882** |
| 130 | fact on Uranus | 0.2976 | 0.0948 | **0.5320** |
| 57 | ct jobs | 0.0112 | 0.0226 | **0.1121** |
| 61 | worm | 0.0104 | 0.1463 | **0.1976** |
| 77 | bobcat | 0.0114 | 0.0406 | **0.1862** |

From Table 4.3 we can observe that for lexical queries that ask for exact or near-exact match, such as "Neil Young" (a musician), "fact on Uranus" and "Texas border patrol", the model with only term-level $S0$ matching outperforms the model with only high level score $S3$. The latter may expand too much the semantic of the query therefore results in poor performance. For example, some of the documents retrieved by Multi-Match-3L_S3 for the query "Neil Young" contain other people named Neil which are irrelevant to this query. Similarly for the query "Texas border patrol", there are documents retrieved by Multi-Match-3L_S3 which contain border patrol of other states than Texas. For the query "fact on Uranus", the model Multi-Match-3L_S3 also retrieves some documents that contain Mars which are not relevant to Uranus.

However, for queries requiring a conceptual match, such as "ct jobs", "worm" and "bobcat", the model with high level score $S3$ outperforms the model with low-level score $S0$, since it can learn high level concept representations and perform matching at this level. For example, among the retrieved documents for the query "bobcat", there are desired documents containing information about bobcat company and bobcat brand tractors. In this case the query has been correctly generalized at conceptual level. For the query "worm", among the retrieved documents, there are desired documents containing the worm's living habits in nature and also desired documents about computer worm virus. For the query "ct jobs", by performing some generalization, the model retrieved desired documents about how to find jobs in Connecticut and those containing the unemployment rate of Connecticut.

We also notice that on the third column, by combining the matching scores of every level, the Multi-Match-Rep model could outperform both the model with $S0$ and $S3$ scores, which demonstrates the power of multi-level matching.

We also analyze some representative queries for interaction-based models and compare the performance of the interaction-based multi-level matching model with some base models in Table 4.4.

**Table 4.4.** NDCG@10 of Representative Queries for Inter Models

| Topic_num | Query | Inter-S0 | Inter-S2 | Inter-MultiMatch |
|---|---|---|---|---|
| 54 | President of the United States | 0.1370 | 0.0367 | **0.1429** |
| 62 | Texas border patrol | 0.2469 | 0.1832 | **0.2577** |
| 153 | Pocono | 0.0609 | 0.0595 | **0.0658** |
| 96 | rice | 0.5435 | 0.6898 | **0.6970** |
| 145 | vines for shade | 0.2225 | 0.3333 | **0.4201** |
| 155 | last supper painting | 0.2240 | 0.3479 | **0.3938** |

For queries which ask for an exact or near-exact match, such as "President of the United States", "Texas border patrol" and "Pocono" (a mountain), the model with only term-level $S_0$ matching outperforms the model with high-level score $S_2$. The latter model may expand too much the semantic of the query, which performs poorly. For example among the documents retrieved for the query "President of the United States" by $Inter - S2$, a number of documents are about presidents of other countries than the United States which are irrelevant to this query. For the query "Texas border patrol", there are also documents retrieved by $Inter - S2$ which contains the name of other states which are irrelevant. For the query "Pocono", some documents retrieved by $Inter - S2$ contains the name of other places which are also irrelevant to this query. These are examples of over generalization.

For conceptual queries such as "rice", "vines for shade" and"last supper painting", the model with high-level score $S_2$ outperforms the model with only term-level score $S_0$, since it captures the high-level interactions required for these queries. For example, the documents retrieved for "last supper painting" by $Inter - S2$ include the desired documents about "description of the last supper painting", "significance of last supper painting in Catholicism". For the query "rice", the model $Inter - S2$ retrieved desired documents about recipes of rice and the nutrition values of rice. In these cases, the query has been correctly generalized at conceptual level.

We also observe in the third column that in both above cases, by dynamically combining the matching scores of the 3 levels of abstraction by a gating mechanism, our Multi-Match-Inter model can outperform each of the single-level base models. These examples show the ability of our model to use the appropriate level(s) of matching depending on the query.

### 4.7.3. Complexity Analysis

In this subsection, we mainly discusses the time complexity of our proposed multi-level convolutional matching models. Both our representation- and interaction-based multi-level matching models are based on Convolution Operations. According to [Vaswani et al., 2017], one 1D convolution layer will cost $O(k*n*d)$ multiplications in inference/test mode or equal amount of gradient operations in training mode, where $k$ is the dimension of the convolved layer vectors, $d$ is the dimension of the input layer vectors, and $n$ is the sequence length (number of terms). According to [He and Sun, 2015], one 2D convolution layer will cost $O(n_{in} * s_f^2 * n_{out} * m_h * m_w)$ operations, where $n_{in}$ is the number of input channels, $s_f$ is the length of filters (in our setting it's squared shape, so is the $s_f^2$), $n_{out}$ is the number of output features maps, and $m_h$, $m_w$ are the height and width of output feature map.

For the representation-based multi-level abstraction convolution model proposed in Section 4.3.2, suppose that the maximum query and document length are $n$ and $m$ respectively, the input embedding dimension is $d$, the hidden size is $h$, and in our setting $h < d$, then for each 1D convolutional layer, to obtain the level-importance score, we need to build local interactions between query and document representations, which is at most of size $n * m$. We also need to perform one layer of 1D convolution for both query and document, which will cost at most $(n + m) * h * d$. Therefore for one layer, it will cost $(n + m) * h * d + n * m$ operations. For $L$ layers, the total complexity will be $O(L((n + m)hd + nm))$. If the hyper-parameters of the network is fixed (i.e. $L$, $h$, $d$ are fixed), then the dominant term will be the $n * m$ and the complexity is bi-linear with respect to query and document length $n$ and $m$.

For the interaction-based multi-level abstraction convolution model proposed in Section 4.4.2, suppose that the maximum query and document length are $n$ and $m$ respectively, the hidden size is $h$, for each convolutional layer, there will be no more than $k$ feature maps and the max filter sizes are $s_f * s_f$, then for each 2D convolutional layer, it will cost at most $k^2 * s_f^2 * n * m$. At each layer $i$, to obtain the level specific score $S_i$, there will be a MLP layer and final score outputting process which will involve two matrix-vector multiplications, the complexity is $n * m * h + h * 1$. So for one layer, it will cost at most $(k^2 * s_f^2 + h) * n * m + h$. With $L$ layers and the final aggregating MLP, the total complexity of the model will be $O(L * ((k^2 * s_f^2 + h) * n * m + h + 1))$. If the hyper-parameters of the network are fixed (i.e. $L$, $s_f$, $k$, $h$ are fixed), then the dominant term will be $n * m$ which is bi-linear with respect to the query and document length $n$ and $m$.

In practical applications, the total time complexity of a retrieval model could be decomposed into two parts: the offline complexity which represents the cost of indexing all documents in the collection in an offline manner and the online complexity which reflects the cost to encode the query and perform retrieval. Representation-based models generally

have low online complexity, since most of the document representation could be learned offline during indexing stages, and the online matching process between query and document representations usually involves simple matching functions such as cosine similarity or dot product. Therefore they are more efficient in serving user's query in a timely fashion. Interaction-based models need to perform complex interactions between query and document early in the lower layers of the retrieval model, and this interaction could only be performed after the query has been known, therefore they usually have higher online complexity but offers better retrieval performance.

## 4.8. Conclusion

Previously, many neural models have been proposed to perform IR tasks. Depending on how the relevance between query and document is calculated, those models could roughly be categorized into representation-based and interaction-based models. Although they have demonstrated their potentials, they have been trained and evaluated on different datasets, which makes it difficult to compare their performance on a fair basis. In this work, we build convolutional models on both schemas, train them and evalutate their performance on the same training and testing collection. Our first goal is to compare the two approaches as fairly as possible. The experimental results clearly show that interaction-based models always outperform representation-based models. The promising performance of interaction-based model is partly attributed to their capacity to learn the local interaction patterns rather than learning global representations in representation-based model.

Moreover, to achieve query-dependent matching, we integrate multi-level matching mechanism into both representation- and interaction-based models and evaluate their performance. The experimental results show that in both schemas, multi-level matching models could outperform models with single-level matching. This result shows the usefulness to design models that can cope with low-level lexical matching as well as high-level semantic matching. In addition, we observe that with multi-level matching mechanism, representation-based model still performs worse than interaction-based model, which further confirms that it is preferable to employ interaction-based models for ad hoc search.

# Chapter 5

# Cross-level Matching Model

## Article Details:

Yifan Nie, Jian-Yun Nie: Cross-Level Matching Model for Information Retrieval. AIRS 2019: 106-117 (Best paper award)

## Context:

At the time of writing this article, many neural retrieval models have been proposed and shown competitive results. In particular, interaction-based models have shown superior performance to traditional models in a number of studies [Guo et al., 2016, Fan et al., 2018]. However, the interactions used as the basic matching signals are between single terms or their embeddings. For instance, in MatchPyramid [Pang et al., 2016a], the basic matching components are term embeddings, and this model employs 2D convolutions to analyze the term-to-term interaction matrix. In DRMM [Guo et al., 2016], the basic matching components are also term embeddings. This model first calculates cosine similarity between each query term embedding with all document term embeddings and then convert the similarities into histogram.

In reality, a term can often match a phrase or even longer segment of text. Sometimes the longer segment of text doesn't even contain exact matched term, but the whole segment could represent a certain concept which could be matched with a single term. In theses cases, in addition to term-to-term local interactions, it is also helpful to add additional matching signals between term and phrase to help the model to cope with the need of term-phrase matching.

## Contributions:

To adapt to this reality, in this section, we propose a Cross-Level Matching Model which enhances the basic matching signals by allowing terms to match hidden representation states within a sentence. A gating mechanism aggregates the learned matching patterns of different matching channels (both term-term and term-phrase) and outputs a global matching score by deciding the importance of each matching channel. Our model provides a simple and effective

way to incorporate both term-to-term and term-phrase matching. Our paper won the best paper award of the Asian Information Retrieval Society 2019 Conference (AIRS2019).

## 5.1. Introduction

Recently many neural models for information retrieval have been proposed and demonstrated their effectiveness [Huang et al., 2013, Shen et al., 2014, Guo et al., 2016, Pang et al., 2016a]. Generally, a neural retrieval model learns either the representations of query and document or the interactions between them through a series of neural layers [Guo et al., 2016, Dehghani et al., 2017]. Those neural layers such as RNN or 1D-convolutions aggregate low-level representations/interactions to produce higher level features [Melamud et al., 2016, Shen et al., 2014]. Afterwards, a matching function is applied on the representations or interaction patterns to produce a matching score.

It has been found [Guo et al., 2016] that interaction-based neural models usually perform better than representation-based models, as we also showed in the previous chapter, because the former can capture more precise matching signals between components (words) in the document and the query. However, in existing interaction-based models, the basic interaction signals from which patterns are extracted are matchings between single words. This may limit the ability of the models to match different, but semantically similar elements. For example, in MatchPyramid [Pang et al., 2016a] and ARC-II [Hu et al., 2014], the basic matching components are term embeddings. These models first build term-to-term local interactions and then analyze the patterns through 2D convolutions. In DRMM [Guo et al., 2016], the basic matching components are also term embeddings. This model first calculates cosine similarity between each query term embedding with all document term embeddings and then converts the similarities into histogram. Finally, several dense layers are employed to analyze the interaction pattern.

However, in reality, a term could often match a phrase or even a longer segment of text, and very often in the segment there could be no exact match of the term but words grouped together, expressing the similar concept. Therefore there does exist the need of term-phrase matching. For instance, for the query "what's the standard barrel length for an AR", it is helpful to match the query phrase "standard barrel length" with the terms "AK47", "Carbine" in relevant documents, since they are all related to guns. For the query "what's the nickname for Nevada", it is also helpful to match the query term "Nevada" to the document phrase "Silver State", because they both refer to the State of Nevada, although the phrase "Silver State" doesn't contain the exact term "Nevada" at all. Similarly, for the query "stainless steel", it is also beneficial to generalize those terms and match with the phrase "corrosion resistance" since it is one of the properties of stainless steel, although there's no exact match between them.

Although previous interaction-based models have demonstrated their potentials, they primarily rely on the matching at term level to generate the interaction patterns. Since the neural layers aggregate low-level terms into higher level representations, it is possible to map the terms inside a phrase into hidden representations and match them with terms, thereby providing additional matching signals to interaction-based models.

Inspired by the above observations, for a neural retrieval model, instead of matching the query and document at term level, a better strategy is to produce the matching signals between the query and document at every layer and across layers. The matching between the term layers will provide valuable information about exact term matches and the matching of synonyms. The cross-matching between the term layer and higher-level layer will provide signals for the matching between term and phrase or segment of larger granularity. Finally the matching between the higher-level layers will generate matching patterns on phrase level or between segments of larger granularity.

To achieve this, in this study, we propose a Cross-level Matching Model for Information Retrieval (CLMM). Our contributions are two-fold: (1) We perform matching between query and document at every representation layer and across layers to produce different matching patterns. (2) We employ a gating mechanism to dynamically aggregate the matching patterns learned by each matching channel to produce a global matching score, thereby paying more attention to more important matching channels. Our experiments on public datasets confirm the effectiveness of cross-level matching over single level matching.

## 5.2. Related Work

Previously many neural IR models have been proposed. Typically, they either rely on learning the representations of query and document separately and then matching the learned query and document representation [Huang et al., 2013, Shen et al., 2014] or building interactions between query and document at an early stage [Guo et al., 2016, Hu et al., 2014, Wu et al., 2017]. The details of these work has been presented in Section 4.2.1.

However, the basic features of query and document employed in the above work are usually the embeddings of single terms. A crucial problem in natural language is that the meaning of a term can often be expressed by a phrase or more complex construct. The basic term-level interactions may miss important matching signals between document and query. Therefore, in this study, we explore the usefulness of matching terms against hidden representations generated within a phrase.

The MACM [Nie et al., 2018] presented in Chapter 4 also considers multiple levels of interaction. In this chapter, we extends it in several aspects: First, for the interaction-based MACM, there is only one matching channel between query and document terms. In this work, Instead of only employing term embeddings, we learn different layers of representations for

the query and document and match the representations (not only term embeddings but also phrase representations) at different levels. Second, we match the query and document not only at the same level (i.e. $(h^{q,(i)}, h^{d,(i)})$), but also across layers (i.e. $(h^{q,(i)}, h^{d,(j)}), i \neq j$). Third, instead of obtaining only a scalar score at each level, we produce a vector representing the matching pattern of each matching channel $(h^{q,(i)}, h^{d,(j)})$ which is expected to be more informative than the scalar value.

The C-KNRM model [Dai et al., 2018] also considers the matching between term and phrase. In this model, the query and document terms are first aggregated into phrase representations by 1D convolutions. Afterwords it performs matching on term-term, phrase-phrase, term-phrase and phrase-term levels between query and documents through interaction matrices. Finally the aforementioned interaction matrices are kernel-pooled and aggregated through a MLP to output a global matching score. Our model is also different from C-KNRM in several aspects: First, instead of kernel pooling, we employ 2D convolutions to analyze the matching patterns of each matching channel. Second, instead of concatenating the learned patterns from each matching channel, we employ a gating mechanism to attentively combine them, thereby paying more attention to important matching channels dynamically.

Recently, with the success of pretrained language models, neural retrieval architectures based on such pretrained language models have also been proposed. In particular, TwinBERT [Lu et al., 2020b] employs BERT [Devlin et al., 2019] as representation learning module to learn the representations of query and document and regards the [CLS] token of query and document as global representations. Afterwards a dot product is applied between query and document global representation to produce a matching score. Although this model demonstrated its potential, it only involves the last layer of BERT representation in the matching process. As mentioned above, the lower layers may also play a role in identifying relevant parts for certain language entities. Therefore it may also be helpful to involve matchings across different BERT layers to serve the need of matching different language entities between query and document. We will demonstrate in our experiments that employing cross-level matching in BERT brings benefits to the retrieval effectiveness.

## 5.3. Cross-level Matching Model

The architecture of our proposed Cross-level Matching Model is illustrated in Fig. 5.1. The model could be roughly divided into 3 parts: representation learning module, cross-level matching channels and gated aggregation module. The representation learning module is responsible of aggregating representations from fine-grained elements (such as terms) to coarse-grained elements (such as phrases). The cross-level matching channels aims to match between different representation elements, such as term-term, term-phrase, and phrase-phrase

and analyze the underlying interaction patterns between them. Finally the gated aggregation module will attentively combine the matching patterns across different matching channels. The details of the components will be presented as follows.

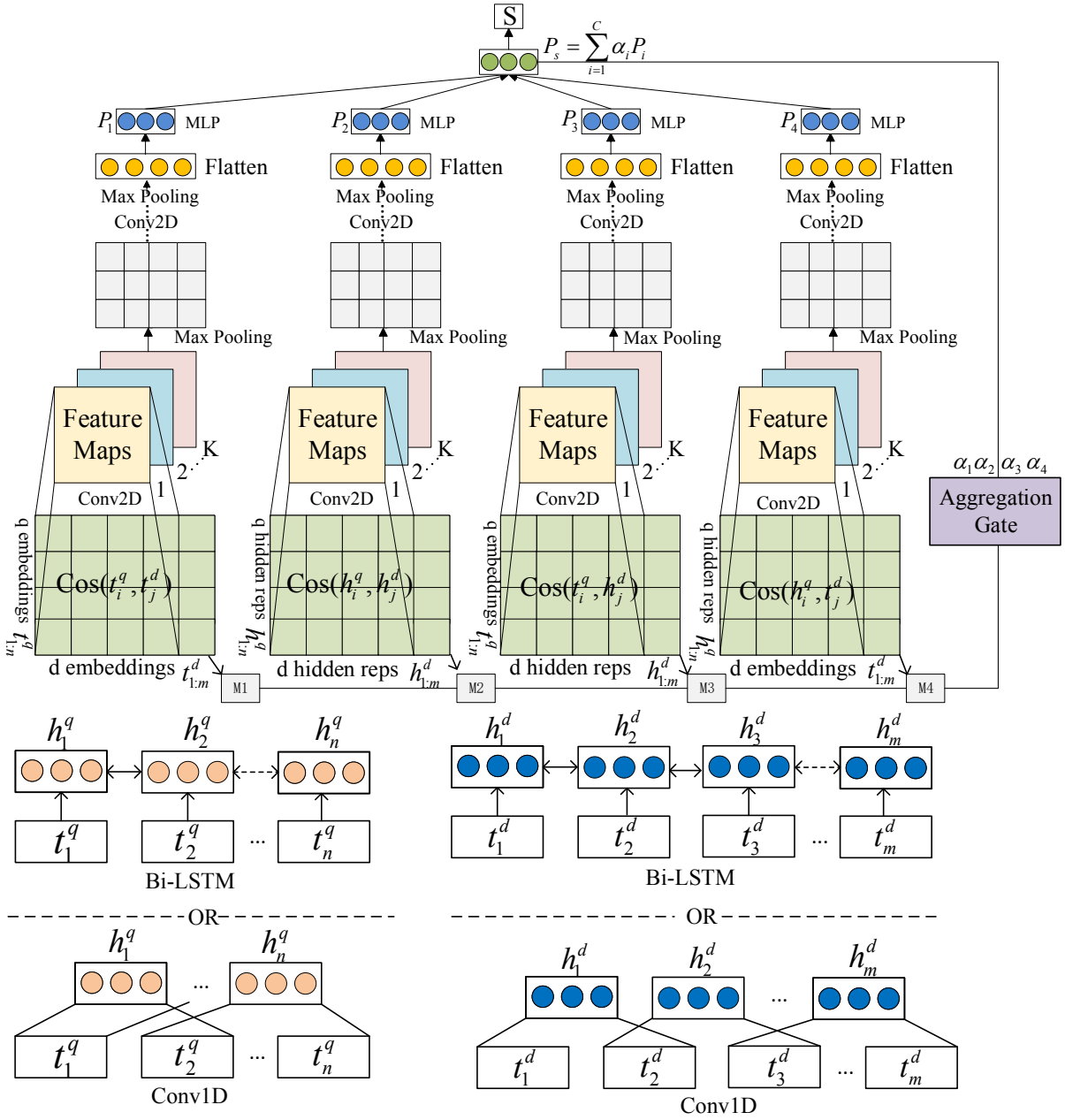



**Fig. 5.1.** Cross-level Matching Model

### 5.3.1. Representation Learning module

Given a query $q$ and a document $d$, we first extract their term embeddings through an embedding lookup matrix and represent them as $q = [t_1^q, t_2^q, ..., t_n^q]$, $d = [t_1^d, t_2^d, ..., t_m^d]$, where $t_i^q$,

$t_j^d$ are the term embeddings for the $i^{th}$ query term and the $j^{th}$ document term. Once the term embeddings are extracted, we learn $L$ layers of representations for the query and document by either Bi-LSTM or 1D-convolutions. For the BiLSTM encoder:

$$h_i^{q,(1)} = BiLSTM(t_i^q, h_{i-1}^{q,(1)}, h_{i+1}^{q,(1)}) \tag{5.3.1}$$

$$h_i^{q,(l)} = BiLSTM(h_i^{q,(l-1)}, h_{i-1}^{q,(l)}, h_{i+1}^{q,(l)}), \;\; l = 2..L \tag{5.3.2}$$

$$h_i^{d,(1)} = BiLSTM(t_i^d, h_{i-1}^{d,(1)}, h_{i+1}^{d,(1)}) \tag{5.3.3}$$

$$h_i^{d,(l)} = BiLSTM(h_i^{d,(l-1)}, h_{i-1}^{d,(l)}, h_{i+1}^{d,(l)}), \;\; l = 2..L \tag{5.3.4}$$

where $h_i^{q,(l)}$ and $h_i^{d,(l)}$ represent the hidden state of the $i^{th}$ query and document term at layer $l$. For the Conv1D encoder:

$$h_i^{q,(1)} = f(W_1^q * [t_{i-w(1)}^q; ..; t_{i+w(1)}^q] + b_1^q) \tag{5.3.5}$$

$$h_i^{q,(l)} = f(W_l^q * [h_{i-w(l)}^{q,(l-1)}; ...; h_{i+w(l)}^{q,(l-1)}] + b_l^q), \;\; l = 2,...,L \tag{5.3.6}$$

$$h_i^{d,(1)} = f(W_1^d * [t_{i-w(1)}^d; ..; t_{i+w(1)}^d] + b_1^d) \tag{5.3.7}$$

$$h_i^{d,(l)} = f(W_l^d * [h_{i-w(l)}^{d,(l-1)}; ...; h_{i+w(l)}^{d,(l-1)}] + b_l^d), \;\; l = 2,...,L \tag{5.3.8}$$

where $h_i^{q,(l)}$ and $h_i^{d,(l)}$ represent the hidden representations of the $i^{th}$ query and document term at layer $l$. $2w(l) + 1$ is the Conv1D window size at layer $l$. $f$ is the $tanh$ activation function.

The above hidden representations can also be learned from more sophisticated representation learning modules such as transformers [Vaswani et al., 2017] or pretrained language model such as BERT [Devlin et al., 2019]. Pretrained language models are shown to be powerful to aggregate contextual information into higher-layer representations through self-attention mechanism, thereby further improving performance in many NLP tasks [Qu et al., 2019, Yates et al., 2021]. To take advantage of the power of the pretrained language models and demonstrate the flexibility of our proposed framework, we also employed the pretrained BERT [Devlin et al., 2019] as an alternative representation learning module.

Similarly to the Bi-LSTM or 1D Convolution representation learning modules, we also first learn a series of representation layers by BERT encoder for each query and document term.

$$h^{q,(1)} = BERT(t^q) \tag{5.3.9}$$

$$h^{q,(l)} = BERT(h^{q,(l-1)}), \;\; l = 2,...,L \tag{5.3.10}$$

$$h^{d,(1)} = BERT(t^d) \tag{5.3.11}$$

$$h^{d,(l)} = BERT(h^{d,(l-1)}), \;\; l = 2,...,L \tag{5.3.12}$$

where $t^q$ and $t^d$ represent sequence of embeddings for the query and document, $h^{q,(l)}$ and $h^{d,(l)}$ represents the $l^{th}$ layer of query and document hidden representations learned by BERT, and $BERT$ represents the BERT module.

### 5.3.2. Cross-level Matching Channels

Once the representations of each layer for the query $[h^{q,(0)},..,h^{q,(L)}]$ and the document $[h^{d,(0)},..,h^{d,(L)}]$ are learned, we perform cross-level matching for each of the combinations of the abstraction layers between query and document by calculating a cosine interaction matrix. As an example, Fig. 5.1 only illustrates the case with Layer 0 (embedding layer) and Layer 1, with in total 4 cross matching channels. The process could be summarized as follows, where $I^{l,m}$ represents the interaction matrix for the $l^{th}$ and $m^{th}$ representation layer of the query and the document.

$$I_{ij}^{l,m} = Cos(h_i^{q,(l)}, h_j^{d,(m)}),\ l = 0,..,L, m = 0,..,L \tag{5.3.13}$$

Once the interaction matrices are built, the interaction patterns are learned through a series of 2D-convolutions. Finally the interaction patterns of the last max-pooled layers are flattened into vectors $Q^{l,m}$ as in [Pang et al., 2016b] and compressed into reduced dimension pattern vectors $P^{l,m}$ by MLP. The process could be summarized as follows, where $C^{k,(l,m)}$ and $M^{k,(l,m)}$ are the $k^{th}$ feature maps of the convolved and maxpooled layer for the matching channel $(l,m)$. $W_{l,m}$ and $b_{l,m}$ are the weight and bias for the dimension reduction MLP.

$$C_1^{k,(l,m)} = Conv2D(I^{l,m}) \tag{5.3.14}$$

$$M_1^{k,(l,m)} = Maxpool(C_1^{k,(l,m)}) \tag{5.3.15}$$

$$C_2^{k,(l,m)} = Conv2D(M_1^{k,(l,m)}) \tag{5.3.16}$$

$$M_2^{k,(l,m)} = Maxpool(C_2^{k,(l,m)}) \tag{5.3.17}$$

$$Q^{l,m} = Flatten(M_2^{l,m}) \tag{5.3.18}$$

$$P^{l,m} = f(W_{l,m}Q^{l,m} + b_{l,m}) \tag{5.3.19}$$

### 5.3.3. Gated Aggregation

To aggregate the matching patterns $P^{l,m}$ learned from each matching channel, a better strategy is to attentively combine them rather than concatenating them together, since for each query-document pair, some matching channels might be more important than the others.

In order to achieve attentive combination of the matching patterns of all matching channels, akin to [Nie et al., 2018] (Chapter 4), we calculate a channel-importance score $M^{l,m}$ for each of the matching channels $(l,m)$ from their interaction matrices $I^{l,m}$. We first take

the maximum value $M_i^{l,m}$ of each row $i$ of the interaction matrix $I^{l,m}$, which represents the strongest interaction between the $i^{th}$ query term / hidden representation and all document terms / hidden representations. Afterwards, we take the average of all these $M_i^{l,m}$ values across the rows to obtain the global maximum interaction of the whole query with respect to the entire document. The process could be summarized as follows.

$$M_i^{l,m} = \max_{j=1..v} I_{ij}^{l,m} \tag{5.3.20}$$

$$M^{l,m} = \frac{1}{u} \sum_{i=1..u} M_i^{l,m} \tag{5.3.21}$$

where $u$, $v$ are the number of rows and columns for the interaction matrix $I^{l,m}$ of the matching channel ($l$,$m$). Intuitively, the higher the value $M^{l,m}$ is, the more we will rely on this matching channel.

Once the channel-importance scores $M^{l,m}$ are produced, we feed them into a softmax gate to get normalized weight for each matching channel.

$$\alpha_{l,m} = \frac{exp(\beta_{l,m} M^{l,m})}{\sum_{r,s} exp(\beta_{r,s} M^{r,s})} \tag{5.3.22}$$

where $\beta_{l,m}$ are learnable scalar parameters, and $M^{l,m}$ are the channel-importance scores for each matching channel. Finally the matching patterns $P^{l,m}$ learned by each matching channel are attentively aggregated with the channel-specific weights $\alpha_{l,m}$ to produce a global pattern vector $P_s$.

Previous studies have also found that non-neural ranking features are useful in certain cases [Severyn and Moschitti, 2015]. Therefore, our model also provides the flexibility to incorporate arbitrary non-neural ranking features $P_{feats}$ into the global pattern vector $P_s$ by concatenating the learned neural matching pattern $P_s$ with $P_{feats}$ to form new hybrid matching pattern $P_g$.

This hybrid matching pattern $P_g$ is then fed into a scoring MLP to produce a global matching score $S$. The whole process could be summarized as follows:

$$P_s = \sum_{(l,m)} \alpha_{l,m} P^{l,m} \tag{5.3.23}$$

$$P_g = Concat([P_s; P_{feats}]) \tag{5.3.24}$$

$$S = f(W_g P_g + b_g) \tag{5.3.25}$$

where $Concat([;])$ is the concatenation operator.

The training is done with a pair-wise loss: given a training example ($Q$,$D_+$,$D_-$), we hope that the score $S(Q, D_+)$ should be higher than the score $S(Q,D_-)$. The loss is defined in Equation 5.3.26, where $\Theta$ includes all trainable parameters of the CLMM Model:

$$L(Q,D_+,D_-;\Theta) = max(0, 1 - (S(Q, D_+) - S(Q,D_-))) \tag{5.3.26}$$

### 5.3.4. Alternative Configurations

To investigate the best strategy to aggregate the learned matching patterns, we also build several alternative combination schemas:

**CLMM-concat:** Once we obtain the flattened vector $Q^{l,m}$ of the last maxpooled layer from each matching channel, we don't compress them into dimension-reduction MLP, but directly concatenate them together, map it into $P_s$ by MLP and optionally concatenate $P_s$ with additional non-neural features $P_{feats}$ to form $P_g$. Finally $P_g$ is fed into the last scoring layer. The whole process could be summarized as follows:

$$P_{cat} = Concat([Q^{0,0}; Q^{0,1}; ...Q^{l,m}; ...Q^{L,M}]) \tag{5.3.27}$$

$$P_s = f(W_{cat}P_{cat} + b_{cat}) \tag{5.3.28}$$

$$P_g = Concat([P_s; P_{feats}]) \tag{5.3.29}$$

$$S = f(W_gP_g + b_g) \tag{5.3.30}$$

where $Concat([;])$ represents the concatenation operation, $f$ is the activation function, $W_{cat}, b_{cat}, W_g, b_g$ are the weights and biases for the MLP and scoring layer.

In this configuration, the matching patterns of all matching channels are mixed together. The combination of learned cross-level matching patterns can be done implicitly. This alternative method is more similar to [Dai et al., 2018]. Compared to the model we propose, the concatenated pattern may be less structured. In reality, the importance of the matching patterns between each cross-matching channel may not be equal. As we stated earlier, for some queries, the matching on term level may be more important while for others, the matching on higher level may be more helpful for the model to judge relevance [Nie et al., 2018]. Therefore this configuration may not allow the model to distinguish between more important matching channels and less important ones, which correspond to different granularity of the matching.

**CLMM-MLP:** After obtaining the flattened vector $Q^{l,m}$ of the last maxpooled layer from each matching channel, we compress them through dimension-reduction MLPs to produce $P^{l,m}$. Then, we concatenate the $P^{l,m}$ together and optionally with additional non-neural features $P_{feats}$ to form $P_g$. Finally $P_g$ is fed into the last scoring layer. The whole process could be summarized as follows:

$$P^{l,m} = f(W_{l,m}Q^{l,m} + b_{l,m}) \tag{5.3.31}$$

$$P_g = Concat([P^{0,0}; P^{0,1}; ...P^{l,m}; ...P^{L,M}; P_{feats}]) \tag{5.3.32}$$

$$S = f(W_gP_g + b_g) \tag{5.3.33}$$

where $Concat([;])$ represents the concatenation operaction, $f$ is the activation function, $W_{l,m}, b_{l,m}, W_g, b_g$ are the weights and biases for the MLP and scoring layer.

This configuration applies MLP to reduce the dimensionality of the flattened vectors of the last maxpooled layers. As we can expect, the pattern vectors obtained from the output of 2D convolutions are often very wide since they incorporate signals from many feature maps. Feeding those wide vectors directly into an aggregation MLP may hinder the model's performance, therefore it is reasonable to first compress them. However, once the output patterns are compressed, this configuration still employs a MLP to mix up the patterns learned from all matching channels, which is similar to CLMM-Concat and may not be the optimal strategy.

**CLMM-MLP-maxpool:** Same as CLMM-MLP, after obtaining the flattened vector $Q^{l,m}$, we first compress them through dimension-reduction MLPs to produce $P^{l,m}$. Then we perform dimension-wise maxpooling on $P^{l,m}$ across all matching channels to obtain $P_s$. Optionally, we could concatenate $P_s$ with non-neural ranking features $P_{feats}$ to form $P_g$ and feed it into the last scoring layer. The whole process could be summarized as follows:

$$P^{l,m} = f(W_{l,m}Q^{l,m} + b_{l,m}) \tag{5.3.34}$$

$$P_s(j) = \max_{l,m}(P^{l,m})(j) \tag{5.3.35}$$

$$P_g = Concat([P_s; P_{feats}]) \tag{5.3.36}$$

$$S = f(W_g P_g + b_g) \tag{5.3.37}$$

where $P_s(j)$ represents the $j^{th}$ element of the vector $P_s$, $Concat([;])$ represents the concatenation operator, $W_{l,m}, b_{l,m}, W_g, b_g$ are the weights and biases for the MLP and scoring layer.

In this configuration, similar to the previous approaches, we first perform dimension reduction on the learned matching patterns by MLP. Afterwards, we perform dimension-wise max-pooling on those reduced matching pattern vectors across all matching channels. Compared to previous configurations, this may be a better strategy, since it selects the max interaction values across all matching channels rather than mixing them up, thus can retain less noise. However, compared to our proposed best combining strategy (gated aggregation), it still may not be optimal, since the max-pooling operation only retains the max interaction signals across different matching channels while discards other lower values. The potential problem of this approach is that the matching signals max-pooled from all channels could be artificially amplified: we could obtain a very strong max-pooled matching pattern vector, while none of the individual channels can yield a strong matching. The problem is related to the implicit assumption that each of the dimensions of the vector represents a separate matching signal, which may be too strong. Indeed, the matching signals are hidden in the vector, and the dimensions in it cannot be interpreted as separable features. By producing the max-pooled vector, we also break the integrity of the matching channels: the whole matching vector of a channel is broken into pieces and combined with those from other

channels. In so doing, the original matching signals encoded in a channel vector may be lost. In contrast, our proposed gating strategy uses each of the matching channels as a whole to determine a score. This may better preserve the integrity of each channel and the matching score is more meaningful. In the next section, we will compare these alternative configurations in experiments.

## 5.4. Experimental Setup and Datasets

In this section, we will present the collection used in this study, the baselines to be compared with our model, evaluation metrics and experimental settings. Those experiments aims to investigate the following research questions:

(1). Does it help to add the matching signals of term-phrase layers in addition to term-term matching signals?

(2). If so, how can we combine the different matching signals effectively?

We will present the collection and evaluation metrics used in this study and the experimental setups in the following subsections.

### 5.4.1. Collection

We employ the MSMARCO, MQ2007, MQ2008 to conduct experiments in this chapter. The details of this collection has been described in Section 3.1. The authors of the MSMARCO collection have already prepared training triples (query, passage, label) in two formats: large (272.2GB) and small (27.1GB). Since the large training set is too large, we employed the small training set which contains 68,750 training queries. As for testing, since this collection is released for a competition and doesn't disclose the evaluation set's judgment, we evaluate our method and all baselines on the development set, which is randomly split into a validation and a test set with a ratio of 1:9. Our validation set contains 698 validation queries and test set contains 6282 test queries. The MSMARCO dataset already contains a baseline run for all validation and test queries. We follow the evaluation protocol to rerank the given baseline run for validation and test queries.

For MQ2007 and MQ2008 datasets, we perform 5-fold cross validation according to the fold split pre-defined by the author of the datasets. On average across the 5 folds, there are 1015, 338, 339 training, validation and test queries in MQ2007 and there are 470, 157, 157 training, validation and test queries in MQ2008. Since the size of MQ2007 is much smaller than MQ2007, following [Pang et al., 2017], we also merge the training folds of MQ2007 into MQ2008 and keep the validation and test folds of MQ2008 unchanged.

It is also worth noting that the sizes of training data of MQ2007 and MQ2008 are much smaller than MSMARCO (1015 and 470 training qureies vs. 68,750 training queries), a huge neural model may not be perfectly trained and we find it beneficial to include additional

non-neural features such as $BM25$ and $LM$ into the model. Therefore during training and evaluation we concatenate $[BM25, LM]$ scores as non-neural features for those MQ datasets as described in the previous section.

### 5.4.2. Baselines

In order to demonstrate the effectiveness of CLMM, we compare our CLMM with the following baselines:

**BM25 [Robertson and Zaragoza, 2009], LM [Zhai and Lafferty, 2001]:** Traditional retrieval models. The BM25 model doesn't involve any term embeddings or neural representations. It is based on exact matched term counts.

**MatchPyramid [Pang et al., 2016a]:** An interaction-based model which doesn't learn hidden representations for query and document. It directly matches the query and document term embeddings by an interaction matrix and learn the matching patterns through 2D-convolutions. This model is equivalent to CLMM with only one matching channel $(H^{q,(0)}, H^{d,(0)})$.

**SMN [Wu et al., 2017]:** SMN first learns hidden representations of query and document through RNNs and matches the hidden states of query and document through 2D-convolutions. This model is equivalent to CLMM with only one matching channel $(H^{q,(1)}, H^{d,(1)})$.

**DRMM [Guo et al., 2016]:** DRMM first builds the interactions between query and document terms by histograms instead of interaction matrices. Afterwards it employs several dense layers to analyze the interaction pattern and output a matching score. It is fully based on matchings between query and document term embeddings, no hidden representations are learned.

**MACM [Nie et al., 2018]** (Chapter 4): A modified version of MatchPyramid which produces a matching score at every level of interaction. The matching scores of all layers are aggregated dynamically through a gating mechanism to produce a global matching score.

**C-KNRM [Dai et al., 2018]:** C-KNRM first builds bi-gram representations on top of query and document terms by 1D convolutions. Afterwards, the matching patterns of 4 matching channels between the uni-grams and bi-grams of query and document are learned through kernel pooling. The learned patterns are concatenated to produce a final matching score.

### 5.4.3. Evaluation Metrics

In this study, we employ Mean Average Precision (MAP), Normalized Discounted Cumulative Gain (NDCGs) as evaluation metrics. They are presented in detail in Section 3.3. In particular we calculate NDCG at cut-off position 1, 3, 10, 20. The MAP is a binary

metric and aims to evaluate the general ad-hoc performance. The NDCGs at different cut-off positions simulate the real search scenario where a user would only want to browse the top-k ranked documents. Additionally, for MSMARCO dataset, we also evaluate on MRR which measures the mean reciprocal rank position of the positive document. This metric is appropriate for a test collection where each query only has one relevant document, which is the case of MSMARCO.

Finally for statistical tests, we perform perform paired t-test with Bonferroni correction [Vialatte and Cichocki, 2008] on our CLMM models with respect to BM25, LM, DRMM, MatchPyramid, SMN, MACM and C-KNRM baselines.

### 5.4.4. Experimental Setup

In this subsection, we will present the detailed experiment setup of our models.

We employ pretrained GloVe.6B.300d embeddings[1] for the Bi-LSTM and Conv1D encoders, and fine-tune them during training. The vocabulary size is set to $400K$ and the embedding dimension is 300.

For the MSMARCO dataset, due to memory limitations, we learn 1 layer of hidden representations for both the BiLSTM and Conv1D representation modules, and perform matching on 4 channels, namely $(H^{q,(0)}, H^{d,(0)})$, $(H^{q,(1)}, H^{d,(1)})$, $(H^{q,(0)}, H^{d,(1)})$, $(H^{q,(1)}, H^{d,(0)})$, where $H^{q,(0)}$, $H^{q,(1)}$ represent the embedding layer and first hidden layer for query and $H^{d,(0)}$, $H^{d,(1)}$ represent the embedding layer and first hidden layer for document. We set the max query and document length to be $n = 20$, $m = 100$ for this dataset. Since the MSMARCO dataset is large, due to hardware limitations we did not run the BERT variants on MSMARCO dataset.

For the MQ2007 and MQ2008 dataset, we also learn the aforementioned 4 matching channels. However, since the document are very long (several thousands terms), it is more appropriate to model the hidden representations by Conv1D and BERT rather than BiLSTM, therefore, for those two datasets we only evaluate our proposed CLMM models with Conv1D and BERT as representation learning modules. Moreover, to tackle the long document problem, for the Conv1D encoder, we cut those documents into passages and match each query with all the passages by cosine similarity as in Equation 5.3.13. We then calculate the passage-level average interaction value by averaging all elements of each passage-level interaction matrix and keep only the top K passages with the highest interaction values to be fed into higher layers of the model. We conducted a preliminary study and set the number of passages to keep $K = 3$. We set the max query and passage length to be $n = 20$ and $m = 100$. For the BERT configurations, due to memory limitations, it is difficult to calculate a passage-wise max interaction since that demands feeding the whole document

[1] https://nlp.stanford.edu/projects/glove/

into the BERT encoder. Therefore, for each document we truncate each document to keep its first 300 terms, which is a common approach to deal with long documents [Dai et al., 2018]. We cross-match the BERT's embedding layer with its last hidden representation layer.

The hidden size for one direction of the BiLSTM is set to 128, so the bi-directional hidden representation's size is 256. The hidden size for Conv1D is set to 256 and the convolutional window size is set to 2.

We limit the max number of 2D convolution layers to 2 and fix the number of 2D feature maps to $[32, 16]$ for the 2 convolution layers. The kernel sizes for the two 2D convolutional layers are set to (3,3) and (5,5). We apply zero padding when needed as it is done in [Pang et al., 2016a]. We fix the pooling size of all max pooling layers to be (2,2), and omit OOV document terms. All hidden sizes in the aggregation components are set to 256.

As for evaluation, for the MSMARCO dataset, since the authors of this dataset have already provided a baseline run for the validation and test queries, we follow this protocol and rerank the provided baseline run during validation and test. For MQ2007 and MQ2008 datasets, the author of the datasets has already split them into 5 folds, therefore we follow this scheme and perform 5-fold cross validation. Following the practice of [Fan et al., 2018, Pang et al., 2017], we do not rerank the validation and test instances over a baseline run, but directly rank the validation and test query-document pairs provided in the respective folds.

## 5.5. Experimental Results

In this section, we will present the experimental results of our proposed Cross-level Matching Model and compare its performance with the baselines. The main experimental results are summarized in Table 5.1, Table 5.2 and Table 5.3.

CLMM-BiLSTM represents the proposed Cross-level Matching Model with BiLSTM as representation learning module, and CLMM-Conv represents Cross-level Matching Model with 1D-convolutions as representation learning module. Conv(1,1) is a model to be compared within the CLMM-Conv group. It employs 1 layer 1D-convolution to learn hidden representations of query and document and match the hidden representations of query and document through 2D-convolutions, and no direct embedding-level matching is involved in this model.

Similarly, on MQ2007 and MQ2008, as mentioned previously, to tackle the problem of long documents, we cut documents into passages and only retain the top K passages with the highest average interaction values (marked with $-kpass$). We also found that for the MQ datasets it is beneficial to include non-neural ranking features BM25 and LM. Those models are marked with $-feats$ in table 5.2 and 5.3. For the BERT-encoder-based models, we match either the query and document embedding layer (models marked with $-$(0,0)), query and

**Table 5.1.** Experimental Results on MSMARCO [1]

| MSMARCO | MAP | NDCG1 | NDCG3 | NDCG10 | NDCG20 | MRR |
|---|---|---|---|---|---|---|
| BM25 | 0.1847 | 0.1214 | 0.1959 | 0.2668 | 0.2942 | 0.1873 |
| LM | 0.1717 | 0.1095 | 0.1828 | 0.2476 | 0.2743 | 0.1739 |
| DRMM | 0.0967 | 0.0535 | 0.0955 | 0.1388 | 0.1582 | 0.0979 |
| MatchPyramid | 0.2053 | 0.1407 | 0.2203 | 0.2940 | 0.3221 | 0.2081 |
| SMN | 0.1941 | 0.1332 | 0.2096 | 0.2744 | 0.3028 | 0.1970 |
| MACM | 0.2091 | 0.1463 | 0.2297 | 0.2970 | 0.3236 | 0.2123 |
| C-KNRM | 0.1883 | 0.1303 | 0.2006 | 0.2657 | 0.2946 | 0.1909 |
| CLMM-BiLSTM-MLP | 0.1931* | 0.1297* | 0.2060* | 0.2746* | 0.3045* | 0.1957* |
| CLMM-BiLSTM-concat | 0.1657* | 0.1046 | 0.1729 | 0.2369* | 0.2658* | 0.1686* |
| CLMM-BiLSTM-MLP-Maxpool | 0.1769* | 0.1191* | 0.1860* | 0.2493* | 0.2791* | 0.1801* |
| CLMM-BiLSTM-Gate | **0.2195*** | **0.1571*** | **0.2406*** | **0.3080*** | **0.3382*** | **0.2223*** |
| Conv(1,1) | 0.1370 | 0.0867 | 0.1398 | 0.1935 | 0.2199 | 0.1391 |
| CLMM-Conv-MLP | 0.2158* | 0.1500* | 0.2352* | 0.3060* | 0.3346* | 0.2186* |
| CLMM-Conv-concat | 0.1758* | 0.1176 | 0.1848* | 0.2485* | 0.2785* | 0.1782* |
| CLMM-Conv-MLP-Maxpool | 0.2156* | 0.1550* | 0.2355* | 0.3040* | 0.3320* | 0.2186* |
| CLMM-Conv-Gate | **0.2310*** | **0.1653*** | **0.2592*** | **0.3244*** | **0.3524*** | **0.2343*** |

document last layer's hidden representations (marked with $-(12,12)$) or cross match BERT's embedding layer and the last layer hidden representations (CLMM-BERT-Cross-feats).

In the following subsections, by comparing the experimental results in Table 5.1, Table 5.2 and Table 5.3, we aim to answer the following research questions:

**1) Is CLMM effective compared to the baselines?**

From Table 5.1, we first observe that with the best aggregating strategy, CLMM-BiLSTM-Gate and CLMM-Conv-Gate outperform all the baselines on all evaluation metrics on MSMARCO. From Table 5.2 and Table 5.3, we also observe that the same holds true on both MQ2007 and MQ2008 datasets: CLMM-kpass-feats-Gate outperforms all baselines on every evaluation metric. This confirms the effectiveness of our proposed CLMM model. If we employ more powerful representation learning modules such as BERT, the performance could be further improved, as the results of CLMM-BERT-Cross-feats indicate in the last section of Table 5.2 and 5.3.

**2) Is cross-level matching more advantageous than single-level matching?**

[1] We perform paired t-test with Bonferroni correction on CLMM models with respect to BM25, LM, DRMM, MatchPyramid, SMN, MACM and C-KNRM baselines. The statistically significant results ($p < 0.05$) are marked with $*$

**Table 5.2.** Experimental Results on MQ2007 [1]

| MQ2007 | MAP | NDCG1 | NDCG3 | NDCG10 | NDCG20 |
|---|---|---|---|---|---|
| BM25 | 0.4584 | 0.3842 | 0.3872 | 0.4305 | 0.4849 |
| LM | 0.4490 | 0.3630 | 0.3720 | 0.4145 | 0.4747 |
| DRMM | 0.4601 | 0.3801 | 0.3921 | 0.4315 | 0.4749 |
| MatchPyramid-kpass-feats | 0.4684 | 0.3907 | 0.4033 | 0.4461 | 0.5010 |
| SMN-kpass-feats | 0.4729 | 0.4046 | 0.4183 | 0.4543 | 0.5080 |
| MACM-kpass-feats | 0.4755 | 0.4075 | 0.4177 | 0.4571 | 0.5123 |
| C-KNRM-kpass-feats | 0.4709 | 0.3993 | 0.4143 | 0.4533 | 0.5050 |
| Conv(1,1)-kpass-feats | 0.4782 | 0.4222 | 0.4258 | 0.4650 | 0.5201 |
| CLMM-kpass-feats-MLP | 0.4816* | 0.4273* | 0.4333* | 0.4656* | 0.5168* |
| CLMM-kpass-feats-Concat | 0.4845* | 0.4141 | 0.4258* | 0.4649* | 0.5170* |
| CLMM-kpass-feats-MLP-Maxpool | 0.4863* | 0.4275* | 0.4308* | 0.4683* | 0.5195* |
| CLMM-kpass-feats-Gate | 0.4945* | 0.4361* | 0.4380* | 0.4777* | 0.5293* |
| BERT-(0,0)-feats | 0.4675* | 0.4033 | 0.4043* | 0.4426* | 0.4959* |
| BERT-(12,12)-feats | 0.4874* | 0.4322* | 0.4362* | 0.4726* | 0.5218* |
| CLMM-BERT-Cross-feats | **0.4961*** | **0.4582*** | **0.4488*** | **0.4839*** | **0.5305*** |

**Table 5.3.** Experimental Results on MQ2008 [1]

| MQ2008 | MAP | NDCG1 | NDCG3 | NDCG10 | NDCG20 |
|---|---|---|---|---|---|
| BM25 | 0.4688 | 0.3524 | 0.4210 | 0.2183 | 0.1115 |
| LM | 0.4569 | 0.3290 | 0.4073 | 0.2122 | 0.1086 |
| DRMM | 0.4631 | 0.3550 | 0.4208 | 0.2190 | 0.1102 |
| MatchPyramid-kpass-feats | 0.4821 | 0.3843 | 0.4382 | 0.2303 | 0.1179 |
| SMN-kpass-feats | 0.4858 | 0.3762 | 0.4426 | 0.2314 | 0.1198 |
| MACM-kpass-feats | 0.4898 | 0.3917 | 0.4481 | 0.2332 | 0.1197 |
| C-KNRM-kpass-feats | 0.4734 | 0.3686 | 0.4360 | 0.2257 | 0.1174 |
| Conv(1,1)-kpass-feats | 0.4875 | 0.3884 | 0.4497 | 0.2288 | 0.1166 |
| CLMM-kpass-feats-MLP | 0.4913* | 0.3899 | 0.4458* | 0.2309* | 0.1167* |
| CLMM-kpass-feats-Concat | 0.4899* | 0.3831 | 0.4473* | 0.2323* | 0.1180* |
| CLMM-kpass-feats-MLP-Maxpool | 0.4948* | 0.3869* | 0.4483* | 0.2333* | 0.1181* |
| CLMM-kpass-feats-Gate | 0.4977* | 0.3987* | 0.4551* | 0.2350* | 0.1207* |
| BERT-(0,0)-feats | 0.4732* | 0.3732 | 0.4286* | 0.2235* | 0.1161* |
| BERT-(12,12)-feats | 0.4945* | 0.4062* | 0.4554* | 0.2362* | 0.1210* |
| CLMM-BERT-Cross-feats | **0.5011*** | **0.4158*** | **0.4704*** | **0.2405*** | **0.1238*** |

In Table 5.1, by comparing the results of MatchPyramid, SMN and CLMM-BiLSTM-Gate, we notice that CLMM-BiLSTM-Gate outperforms both MatchPyramid and SMN. MatchPyramid doesn't learn hidden representations of the query and document and directly matches query and document term embeddings. It is equivalent to CLMM with one matching channel $(H^{q,(0)}, H^{d,(0)})$. In this configuration, the model is good at capturing the matching signals at the term level but lacks the capability to capture semantic matching at higher abstraction levels. SMN learns 1 layer of hidden representations of the query and document and matches the hidden representations of the query and document. It is equivalent to CLMM with one matching channel $(H^{q,(1)}, H^{d,(1)})$. In this configuration, the model focuses on the matchings in the hidden semantic space but may ignore some important matching signals at term level. If we consider the configuration of Conv1D as representation module, the same holds true for the comparison between MatchPyramid, Conv(1,1) and CLMM-Conv-Gate.

As for the MQ datasets, by comparing the results of MatchPyramid-kpass-feats, Conv(1,1)-kpass-feats, and CLMM-kpass-feats-Gate in Table 5.2 and Table 5.3, we observe that the same holds true on the MQ datasets. The same holds true for the comparison between BERT-(0,0)-feats, BERT-(12,12)-feats and CLMM-BERT-Cross-feats, where the cross matched model CLMM-BERT-Cross-feats outperformed both the model matching only query and document embeddings (BERT-(0,0)-feats) and the one matching only query and document's last hidden representation layers (BERT-(12,12)-feats)

From the above comparisons, we can conclude that using multiple matching channels is clearly better than using any of the single-level matching channels. This confirms our earlier intuition that various language entities may express similar meanings and this should be incorporated into the matching function between query and document.

**3) What's the best aggregation strategy?**

We observe that among the different aggregation strategies, the gated aggregation works the best on all the three datasets. This confirms that the aggregation of different matching channels should be based on their utility for the specific document-query pair. The score-guided gating mechanism which dynamically weighs the contributions of different matching channels and combines them, is an adequate means. Both alternative models CLMM-Concat and CLMM-MLP use concatenation of the matching patterns of different channels. In this case, the pattern vectors from different matching channels are treated without differentiation. However, queries are of different types. For some queries, a term-level match channel is more important while for others, a match on the phrase level is more appropriate. This possibly explains why the concatenation strategy may not be the optimal approach.

The CLMM-MLP-Maxpool takes the dimension-wise max across the pattern vectors of all matching channels. The resulting matching vector may not be meaningful and does not preserve the integrity of the matching channels, as we explained earlier.

The CLMM-Gate attentively combines the learned interaction patterns from all matching channels according to the importance of each matching channel. It allows each of the matching channels to operate completely before aggregating them. The matching score produced in this way makes more sense than that based on max-pooled vector.

**4) Combination with non-neural features**

It has been shown that the combination of non-neural features with neural components could help improve the performance of a neural retrieval model [Severyn and Moschitti, 2015]. While the neural features focus on the representations or interaction intensities between each query and document, those additional non-neural features could be some scores involving global statistics over the entire collection.

For the MQ datasets, we find it beneficial to employ BM25 and LM scores as the additional non-neural features in ranking. Particularly, the BM25 score contains the inverse document frequency (IDF) for each term which characterizes the rarity of the term in the whole collection and the LM score contains the background collection smoothing term. Those additional features are believed to be able to complement the model with some global statistical information, therefore help to improve performance. In fact, we performed a preliminary study with Conv1D based CLMM-Conv models while removing the non-neural ranking features (BM25 and LM scores), the experimental results on MQ2007 and MQ2008 are presented in Table 5.4.

**Table 5.4.** Alternative Configurations of CLMM without Non-Neural Features

| MQ2007 | MAP | NDCG1 | NDCG3 | NDCG10 | NDCG20 |
|---|---|---|---|---|---|
| MatchPyramid-kpass | 0.4302 | 0.3529 | 0.3552 | 0.3932 | 0.4543 |
| SMN-kpass | 0.4294 | 0.3508 | 0.3598 | 0.3966 | 0.4560 |
| CLMM-kpass-Gate | 0.4406 | 0.3691 | 0.3721 | 0.4127 | 0.4691 |
| MQ2008 | MAP | NDCG1 | NDCG3 | NDCG10 | NDCG20 |
| MatchPyramid-kpass | 0.4415 | 0.3407 | 0.3979 | 0.2064 | 0.0972 |
| SMN-kpass | 0.4396 | 0.3424 | 0.3886 | 0.1994 | 0.0922 |
| CLMM-kpass-Gate | 0.4577 | 0.3504 | 0.4151 | 0.2150 | 0.1037 |

From Table 5.4, we can observe that when the non-neural ranking features are removed, our proposed cross-level matching model (CLMM-kpass-Gate) still outperforms the model that only matches query and document embedding layer (MatchPyarmid-kpass) and the one that only matches query and document hidden representations (SMN-kpass). However the performance of both CLMM and the baselines drop below the performance of a traditional model such as BM25 (reported in Table 5.2 and Table 5.3). This could possibly be explained by the size of the training data: MQ2007 and MQ2008 only have 1015 and 470 training

queries respectively (whereas MSMARCO has 68,750 training queries), while for the training of a deep model, it may need much more training data to achieve better generalization to unseen test instances. Therefore, for those smaller datasets, it may be beneficial to include those traditional, non-neural ranking features (such as BM25 and LM scores) which involve some global statistics over the entire collection.

## 5.6. Discussions and Analysis

In this section, in order to better understand the advantage of cross-level matching over tradition local matching or same-level matching, we will discuss the experiment results, present some case studies based on some typical queries in the test collection and give the complexity analysis of our proposed Cross-level Matching Model.

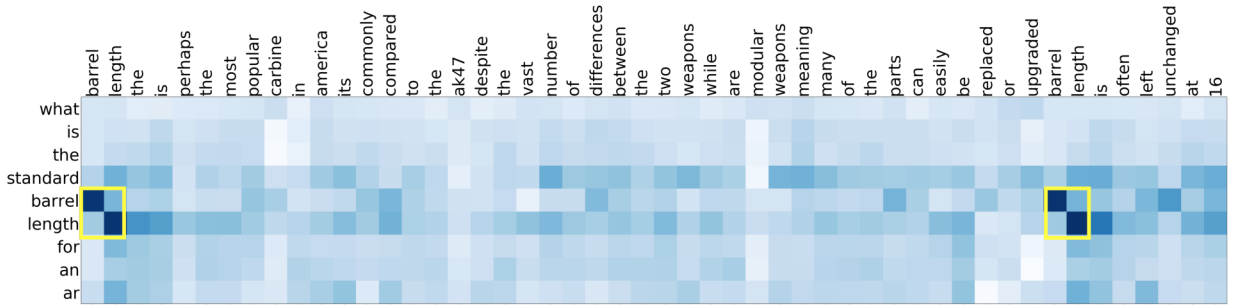
### 5.6.1. Case Studies

In order to further understand the advantage of matching different representation layers between the query and document and the roles of different matching channels, we take a representative test query "what is the standard barrel length for an AR" (a type of gun), and a relevant passage to this query. We visualize the interaction matrices $I^{0,0}$, $I^{1,1}$, $I^{1,0}$ and $I^{0,1}$ of CLMM-BiLSTM-Gate for the matching channels $(H^{q,(0)}, H^{d,(0)})$, $(H^{q,(1)}, H^{d,(1)})$, $(H^{q,(1)}, H^{d,(0)})$ and $(H^{q,(0)}, H^{d,(1)})$ respectively in Fig 5.2.

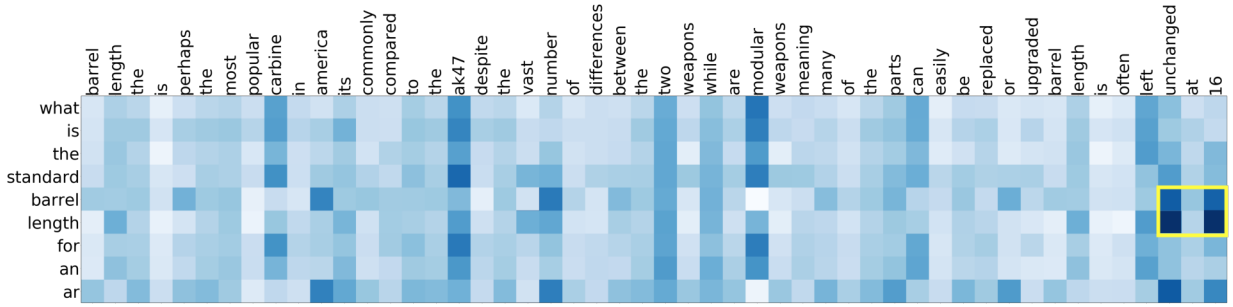
Note that the interaction matrix $I^{0,0}$ captures the matching intensities between query and document term embeddings, $I^{1,1}$ captures the matching intensities between the hidden representations of query and documents, which are believed to represent the semantics in a phrase, and $I^{1,0}$ represents the matchings between the query hidden representations and the document terms, which serves the need of term-phrase matching.

From Fig 5.2a, we can observe that the term embedding matching channel $I^{0,0}$ is responsible to identify the exact-matched terms. In this example, the terms "barrel length" which occur in both the query and document have very high matching intensities. This shows that the channel $I^{0,0}$ which is capable to capture exact matches, is very useful to fulfill the need for lexical matches.

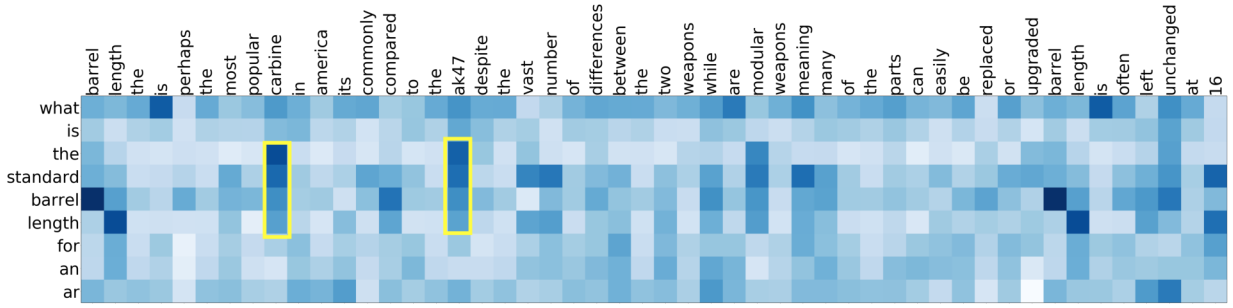
From Fig 5.2b, we notice that the hidden representation matching channel $I^{1,1}$ is responsible to match in the latent semantic space. In this case, query terms "barrel length" and relevant document terms "unchanged at 16" are mapped into hidden representations and matched with high intensities, which helps the model to successfully identify the answer to this query. This demonstrates that the channel $I^{1,1}$ is essential to generalizing query and document terms and performing matching in the latent semantic space.



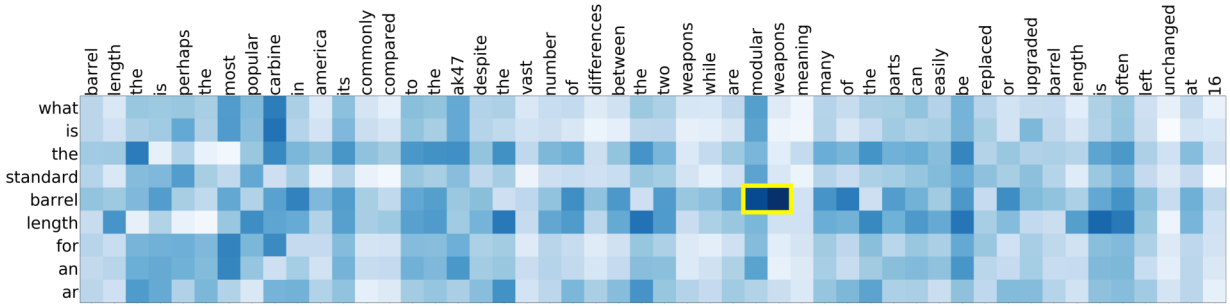

**(a)** Interaction Matrix $I^{0,0}$ of query and document embeddings




**(b)** Interaction Matrix $I^{1,1}$ of query and document hidden representations




**(c)** Interaction Matrix $I^{1,0}$ of query hidden representations and document embeddings




**(d)** Interaction Matrix $I^{0,1}$ of query embeddings and document hidden representations

**Fig. 5.2.** Visualization of Matching Matrices, MSMARCO

Fig 5.2c shows that the cross-level matching channel $I^{1,0}$ (query hidden representation with document term embeddings) is also helpful. In this case, the model successfully associates the document terms such as "Carbine" and "AK47" with hidden representations of query phrase "standard barrel length". Since the semantics of "barrel length" are related to

guns, matching them with guns such as "Carbine" and "AK47" will help the model to find the relevant document containing those terms.

Fig. 5.2d illustrates that the cross-level matching channel $I^{0,1}$ (query term embeddings with document hidden representations) also plays a role to match query terms with relevant document phrases. In this case, the query term "barrel" is matched with the document phrase "modular weapons" with high intensity. Since the barrel is a part of a modular weapon, this matching will boost the model's capability to find relevant document to this query.

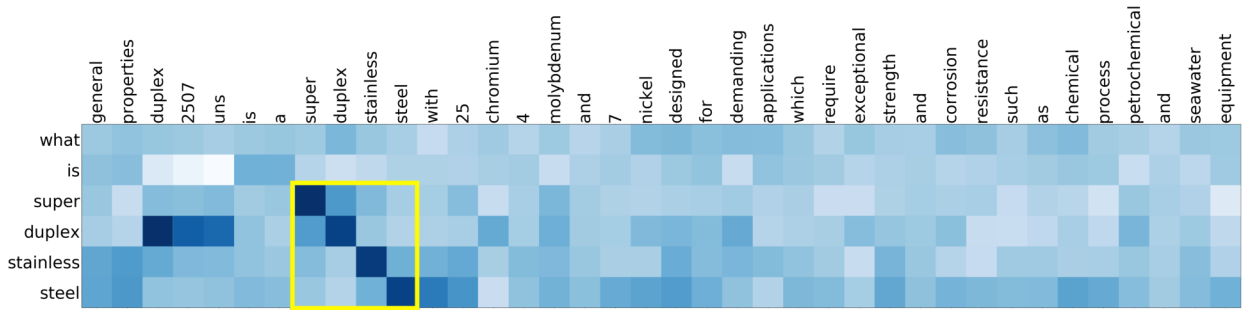
We also take another representative test query "what is super duplex stainless steel" and a relevant passage to this query. We plot the interaction matrices $I^{0,0}$, $I^{1,1}$, $I^{1,0}$ and $I^{0,1}$ of CLMM-BiLSTM-Gate for the matching channels $(H^{q,(0)}, H^{d,(0)})$, $(H^{q,(1)}, H^{d,(1)})$, $(H^{q,(1)}, H^{d,(0)})$ and $(H^{q,(0)}, H^{d,(1)})$ respectively in Fig 5.3.

From Fig 5.3a, we can observe again that the term embedding matching channel $I^{0,0}$ is responsible to identify the exact-matched terms. In this example, the single terms "super", "duplex", "stainless", "steel" which occur in both the query and document have very high matching intensities. This shows that the channel $I^{0,0}$ which is capable to capture exact matches, plays an important role for lexical matches.

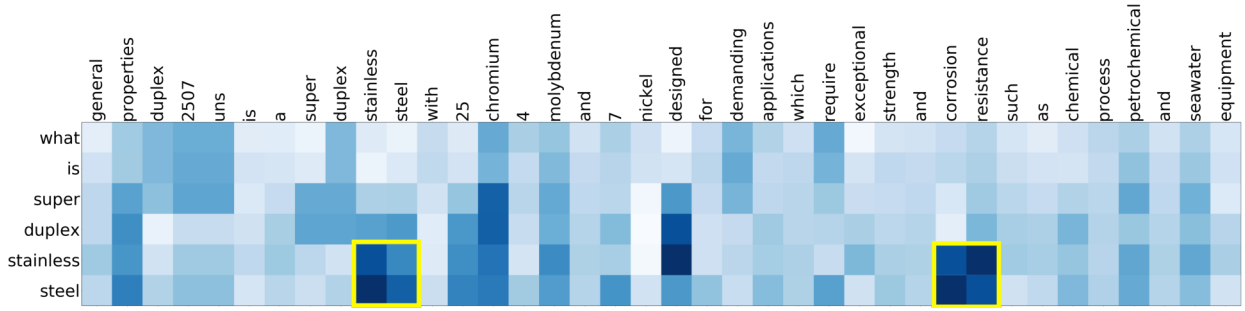
From Fig 5.3b, we notice that the hidden representation matching channel $I^{1,1}$ is responsible to match in the latent semantic space. In this case, query phrase "stainless steel" is matched to the same phrase appearing in the document passage and it is also generalized to match the document phrase "corrosion resistance", since "corrosion resistance" is one of the properties of stainless steel and is highly relevant to it. Therefore the hidden representation channel helps the model to generalize the query semantics and match them in the latent space with document representations.

Fig 5.3c shows that the cross-level matching channel $I^{1,0}$ (query hidden representation with document term embeddings) is also helpful. In this case, the model successfully associates the query phrase "stainless steel" with the document terms such as "chromium" , "nickel" and "corrosion". Since chromium and nickel are essential component of stainless steel and corrosion is also related to stainless steel, this channel serves the need of term-phrase matching and helps the model to match relevant document terms with query phrase.

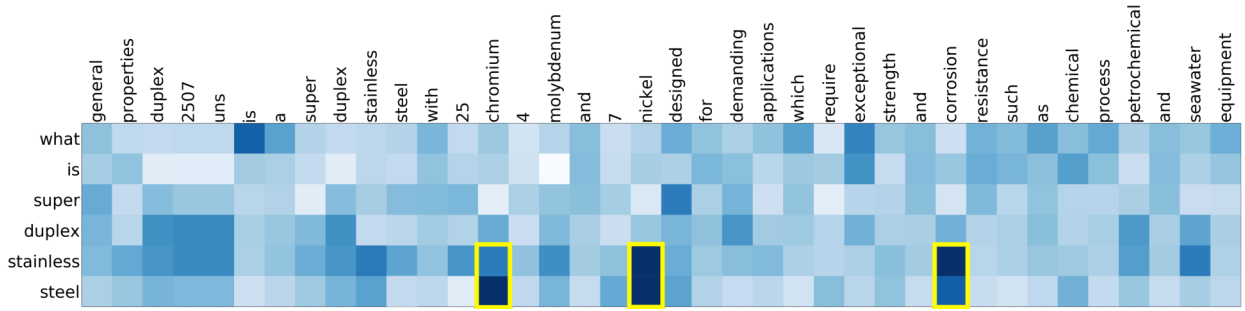
Finally Fig. 5.3d illustrates the matching patterns in the cross-level matching channel $I^{0,1}$ (query term embeddings with document hidden representations). In this case, we notice that the model successfully matches the query term "stainless" with document phrase "corrosion resistance", which is a property related to stainless. Moreover, the model also matches the query term "steel" with the document phrase "exceptional strength" with high interaction intensities, since steel does present the property of exceptional strength. This example demonstrates that the matching channel $I^{0,1}$ also plays a helpful role to match query with document containing relevant phrases.



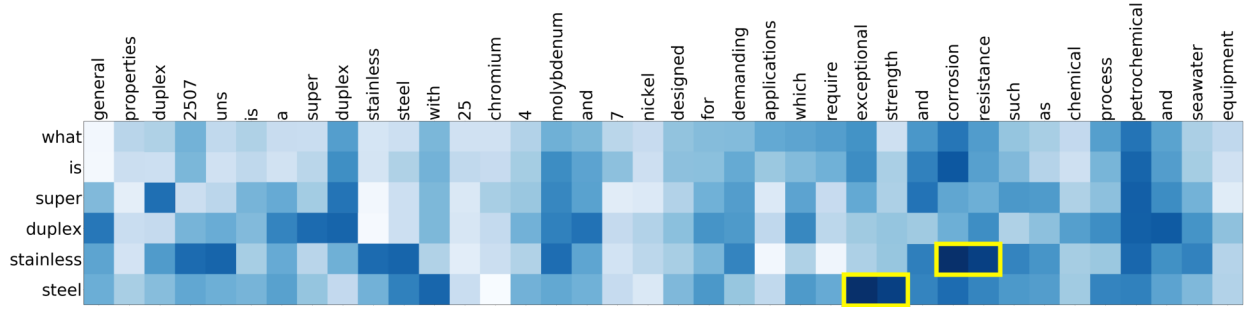

**(a)** Interaction Matrix $I^{0,0}$ of query and document embeddings




**(b)** Interaction Matrix $I^{1,1}$ of query and document hidden representations




**(c)** Interaction Matrix $I^{1,0}$ of query hidden representation and document embeddings

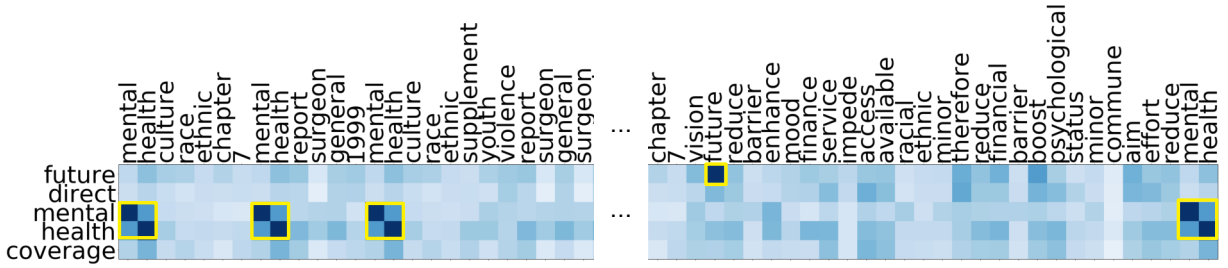



**(d)** Interaction Matrix $I^{0,1}$ of query embeddings and document hidden representation

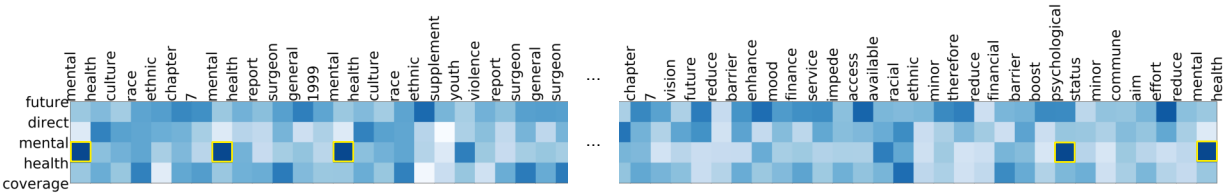
**Fig. 5.3.** Visualization of Matching Matrices, MSMARCO

To illustrate the potential of our CLMM model on different genre of collections, we also take another representative test query from MQ2007 Dataset: "future direct mental health coverage" and a relevant document and visualize the interaction matrices $I^{0,0}$, $I^{1,1}$, $I^{1,0}$ and

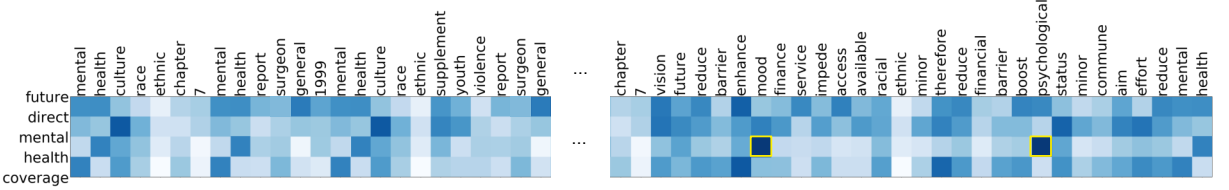
$I^{0,1}$ of CLMM-kpass-feats-Gate for the matching channels $(H^{q,(0)}, H^{d,(0)})$, $(H^{q,(1)}, H^{d,(1)})$, $(H^{q,(1)}, H^{d,(0)})$ and $(H^{q,(0)}, H^{d,(1)})$ respectively in Fig 5.4.



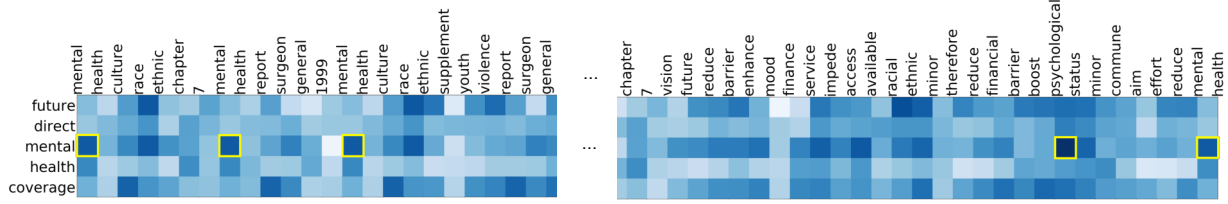

**(a)** Interaction Matrix $I^{0,0}$ of query and document embeddings




**(b)** Interaction Matrix $I^{1,1}$ of query and document hidden representations




**(c)** Interaction Matrix $I^{1,0}$ of query hidden representation and document embeddings




**(d)** Interaction Matrix $I^{0,1}$ of query embeddings and document hidden representations

**Fig. 5.4.** Visualization of Matching Matrices, MQ2007

From Fig 5.4a, we can observe the same phenomenon that the term embedding matching channel $I^{0,0}$ is responsible to identify the exact-matched terms. In this example, the exact-matched terms "mental" and "health" in both the query and document have very high matching intensities. This shows again that the channel $I^{0,0}$ is very helpful to fulfil the need for lexical matches.

In Fig 5.4b, it is worth noting that this is the matching channel $I^{1,1}$ between hidden representations of query and document learned by Conv1D with convolution window size set to 2. Therefore each square represents the matching intensity of fused query and document bi-gram and the individual terms are placed at the edge of each square. Again, we notice that this hidden representation matching channel is responsible to match in the latent

semantic space. In this case, query phrase "mental health" is not only matched with its exact appearance in document but also matched with relevant phrase such as "psychological status". This helps the model to boost the matching score of this relevant document. This demonstrates that the channel $I^{1,1}$ is essential to generalizing query and document terms and performing matching in the latent semantic space.

Fig 5.4c shows the matching patterns in the cross-level matching channel $I^{1,0}$ (query hidden representation with document term embeddings). In this case, since we learn the query hidden representation with Conv1D with convolution window size set to 2 and the document representations remain to be term embeddings, each square represents the matching intensity between query bi-gram and document term and the query terms are placed at the edge of each square. We notice again that the model successfully associates query phrase "mental health" with relevant document terms such as "mood" and "psychological". Therefore, this matching channel will help the model to match query phrase with relevant document terms.

Finally Fig. 5.4d illustrates the matching patterns in the cross-level matching channel $I^{0,1}$ (query term embeddings with document hidden representations). As we learn the document hidden representations by Conv1D with convolution window size set to 2 while the query representations remain to be term embeddings, each square represents the matching intensity between query term and document bi-gram and the document terms are labeled at the edge of each square. In this case, we notice that the model not only successfully matches the query term "mental" with document phrase "mental health", but also matches it with phrase "psychological status" in the document. This example demonstrates that the matching channel $I^{0,1}$ also plays a helpful role to match query with document containing relevant phrases.

In conclusion, through the above analysis of representative queries and documents from collections of different genre, we have a better understanding of the function of different matching channels in our proposed CLMM model. Both the term matching channel $I^{0,0}$, hidden representation matching channel $I^{1,1}$ and cross-level matching channels $I^{1,0}$ or $I^{0,1}$ have their own contributions to identify either an exact match, a semantic match or a term-phrase match. Coupled with an attentive gating mechanism, they work collaboratively to boost the performance of the model to find the relevant documents associated with a given query.

### 5.6.2. Complexity Analysis

In this subsection, we analyze the time complexity of our proposed Cross-level Matching Model. Our representation learning modules are based on either 1D-CNN or BiLSTM. According to [Vaswani et al., 2017], one 1D convolution layer will cost $O(k * n * d)$ multiplications in inference/test mode or equal amount of gradient operations in training mode,

where $k$ is the dimension of the convolved layer vectors, $d$ is the dimension of the input layer vectors, and $n$ is the sequence length (number of terms). According to [Brahma, 2018], one Bi-direction LSTM layer will cost linear time with respect to sequence length $n$, i.e. its complexity is $O(n)$. Our matching channels are built with 2D convolutions. According to [He and Sun, 2015], one 2D convolution layer will cost $O(n_{in} * s_f^2 * n_{out} * m_h * m_w)$ operations, where $n_{in}$ is the number of input channels, $s_f$ is the length of filters (in our setting it's squared shape, so is the $s_f^2$), $n_{out}$ is the number of output features maps, and $m_h$, $m_w$ are the height and width of output feature map.

For our Cross-level Matching Model in Fig. 5.1, suppose that the maximum query and document length are $n$ and $m$ respectively, the input embedding dimension is $d$, the hidden size is $h$, and in our setting $h < d$. if we employ the 1D-CNN as representation learning modules, the representation learning would cost $(n + m) * h * d$ operations. If we employ BiLSTM as representation learning modules, the representation learning would cost $\beta*(n+n)$ operations, where $\beta$ is a constant related with the configuration of the BiLSTM cells.

For each of the matching channels, and for each convolutional layer, suppose that there will be no more than $k$ feature maps and the max filter sizes are $s_f * s_f$, then for each 2D convolutional layer, it will cost at most $k^2 * s_f^2 * n * m$ operations. The construction of the interaction matrix will cost at most $n * m$ operations, and there would be an extra $n * m * h + h * 1$ operations for the final MLP layer. Therefore, for each channel there would be $2 * (k^2 * s_f^2 + h + 1) * n * m + 2h$ operations.

For a total of 4 channels, if we employ 1D convolution as representation learning module, the total complexity will be $O(8(k^2 * s_f^2 + h + 1) * n * m + 4(n + m) * h * d + 8h)$. If we employ BiLSTM as representation learning modules, the total complexity will be $O(8(k^2 * s_f^2 + h + 1) * n * m + 4\beta * (n + m) + 8h)$. In either case, if the hyper-parameters of the network are fixed (i.e. $d$, $s_f$, $k$, $h$, $\beta$ are fixed), then the complexity of the model is dominated by $n * m$ which is bi-linear to the length of query $n$ and document $m$.

## 5.7. Conclusion

Previously many neural IR models have been proposed and demonstrated their potentials, however, most existing neural retrieval models rely on either the representations or interactions of only one layer to perform matching. In reality, the relevant semantics of a term could also be expressed in phrases. Therefore there exists the need for term-phrase matching. To address this challenge, in this chapter, we propose a Cross-level Matching Model for Information Retrieval, which learns multiple query and document representations and matches them across all possible layers to produce different matching patterns. A channel-aware gating mechanism aggregates the matching patterns of each matching channel attentively to

produce a global matching score. Experiments on public datasets confirm the advantage of employing multiple matching channels to enhance the basic term matching signals.

# Chapter 6

# Integrated Learning of Features and Ranking Function

## Article Details:

Yifan Nie, Jiyang Zhang, Jian-Yun Nie: Integrated Learning of Features and Ranking Function in Information Retrieval. ICTIR 2019: 67-74

## Context:

At the time of writing this article, there has been a flourishing trend of employing neural models for IR tasks. Those models could be roughly divided into representation-based models [Huang et al., 2013, Shen et al., 2014], interaction-based models [Pang et al., 2016a, Guo et al., 2016] and hybrid models (combination of the two schemes) [Mitra et al., 2017]. Those deep IR models are good feature-learning models thanks to the application of neural networks, yet most of those existing neural models use a simple pair-wise hinge loss to train.

When it comes to learning a better ranking function, the learning-to-rank framework (L2R) such as LambdaRank provided a better solution. In particular, the LambdaRank could learn a better ranking function by optimizing the ranking metrics such as NDCG. However, we still observe a gap between deep IR models and L2R framework: the traditional L2R framework is feature-based and the input features are usually fixed, hand-crafted features.

Therefore it is natural to raise the question: Is it possible to take advantage of both the feature-learning power of neural models and the good ranking function learned by L2R framework by combining them together? If yes, how could we do so? In the previous chapter, we also explored the utilization of manually defined ranking features and we found this very useful. So, do the manually defined features continue to bring benefits within the L2R framework?

## Contributions:

In order to take advantage of both the feature-learning power of the neural models and the good ranking function learning by learning-to-rank framework, in this chapter, we propose

an integrated learning framework based on learning-to-rank to learn both neural features and the L2R ranking function simultaneously. The framework also has the flexibility to integrate arbitrary traditional features. Our experiments on public datasets confirm that such an integrated learning strategy is better than separate learning of features and ranking function, and integrating traditional features can further improve the results.

## 6.1. Introduction

We have described the existing neural approaches based on representations or interactions in Section 4.2.1. Intuitively, representation-based models can create more abstract representations about the contents, thereby enable some generalization so that similar contents can be matched. They are more appropriate to cope with conceptual queries which require some generalization. For example, for queries like "small business statistics", the documents containing "small company statistics" or "micro corporate statistics" may also be relevant. On the other hand, interaction-based models first build local term-to-term interactions by applying a similarity function and the learning process focuses on interaction patterns allowing to match the query and documents. Therefore, they are more able to deal with lexical queries which require exact term match between query and document. Queries that contain named entities such as "distance between Grand Canyon and Phoenix" fall into this category. In this case, it is preferred that the retrieved documents contain Grand Canyon and Phoenix, not elsewhere.

The above examples illustrate the respective strength of the two approaches, therefore it is natural to combine the above two approaches into an integrated approach so as to take advantage of both approaches. However, such a combined approach has not been extensively studied. The Duet Model [Mitra et al., 2017] is an exception which combines the two approaches by linearly adding the ranking scores of two separate models and learns the ranking function by maximizing the probability score of positive document over negative documents for a given query. Despite the fact that Duet improved the retrieval effectiveness, we believe that both approaches can be better integrated. In particular, the way the two components interact with each other should be learned, and a better ranking function could be employed.

As stated before, neural models are capable of learning powerful features, but the existing neural models only employ simple ranking functions. Learning-to-rank (L2R) framework such as LambdaRank provides a better approach to learn a ranking function, but the features employed in traditional L2R approaches are usually hand-crafted fixed features. They are less expressive than the neural features learned by a neural model. Therefore it is natural to raise the question: Is it possible to combine the neural representation and interaction feature

learning under the L2R framework to take advantage of both the powerful neural features and better ranking function learned by L2R?

To investigate this question, In this chapter, we propose a new general approach to combine representation-based, interaction-based models and arbitrary additional traditional features through learning-to-rank: the representation- and interaction-based models will generate a set of features that are fed into a L2R layer, and the latter will learn an appropriate ranking function based on the features. Different from the traditional L2R methods, the features used are also learned at the same time as the ranking function. Therefore, the approach we propose integrates both feature learning and ranking function learning. Compared to separate feature learning and ranking function learning, an integrated learning framework has a clear advantage: the features can adapt depending on how they are used in the ranking function, and the ranking function also adapts depending on the features at hand. Both the features and the ranking function are learned to maximize the final objective of document ranking.

The utilization of representation and interaction features in traditional L2R is not new: Most traditional L2R approaches utilize both categories of features [Qin and Liu, 2013, Liu, 2009]. It has been shown in many experiments that both types of feature are useful and they are complementary. Our approach follows the same principle, but within the framework of neural networks, and by incorporating neural feature learning as well. The incorporation of optional non neural features has also been applied in recommendation systems. For example, in Wide&Deep [Cheng et al., 2016], a series of attribute features of item are incorporated under the "wide" part of the model , whereas the "deep" part are responsible to learn neural features of user-item interactions.

To summarize, this work proposes a method to combine the representation- and interaction-based neural approaches within the learning-to-rank framework, in which both feature learning and ranking-function learning are conducted end-to-end. We call it Integrated Learning Model (ILM). Our contribution is three-fold: (1) We combine representation-based and interaction-based neural approaches in a flexible learning-to-rank framework. (2) We integrate feature learning with L2R ranking function learning, which are trained end-to-end simultaneously. (3) We show that the proposed model can significantly outperform the existing neural models on Million Query datasets, and that integrating traditional features can further improve the performance.

## 6.2. Related Work

### 6.2.1. Neural IR Models

We have presented previous representation-based and interaction-based models in Section 4.2.1. As we mentioned, representation-based and interaction-based neural models have their respective strengths. To take advantage of the strength and focus of both matching mechanisms, the original Duet Model [Mitra et al., 2017] combines both a representation-based model and an interaction-based model, which generates two independent relevance scores. These scores are then added to produce a final score:

$$R_{Duet}(q, d) = R_{rep}(q, d) + R_{inter}(q, d) \tag{6.2.1}$$

To provide more informative matching signals, a recent Updated Duet Model [Mitra and Craswell, 2019] produces matching pattern vectors of the two sub-models and combines them through MLPs instead of adding the matching scores. The extended Duet model goes in the same direction as our ILM model, but is done in parallel to our work.

Although Duet Model and its variants yielded better results than the representation-based and interaction-based models separately, we can see several limitations. First, in the local interaction model, only exact match function is employed, which does no allow matching similar terms. It might be more reasonable to employ a more flexible function to capture not only exact matches but also synonyms matched at this local position. Second, the ranking function employed in Duet is a softmax probability over scores of positive document and negative documents for a given query. It is better to optimize the final ranking objective for ranking tasks as shown in [Burges, 2010, Tan et al., 2013].

### 6.2.2. Learning-to-rank

Learning-to-rank (L2R) is a general ranking framework for IR which yielded state-of-the-art results [Liu, 2009]. According to how the ranking model is trained, L2R algorithms could be categorized into 3 types: point-wise approaches, pair-wise approaches and list-wise approaches [Liu, 2011]. In particular, pair-wise approaches care about the relative preference between two documents given a query. A popular and effective pair-wise L2R model is LambdaRank [Burges, 2010]. LambdaRank tries to push relevant documents upwards and non relevant documents downwards in a ranked list during training by optimizing the NDCG metric [Wang et al., 2013]. As the training objective function directly corresponds to the evaluation metric, LambdaRank can outperform similar ranking models trained with a pairwise hinge or cross-entropy loss [Burges, 2010]. In [Burges, 2010], an MLP is used to act as ranker. A L2R model works with a set of features extracted from a document-query pair such as document and query length, term frequencies, and different matching scores between

them. It has been found that both the features relating to the query and to the documents (contents), as well as the features relating to their interactions (matching scores or patterns) are important. This observation motivates us to combine both representation-based and interaction-based neural features in an integrated framework.

L2R has been recently adapted to the neural model context. In [Ai et al., 2018], a Group Scoring Function Model (GSF) is proposed. For a given query $q$ and a list of document $[d_i, ..., d_n]$, the model considers each possible group $(q, d_i, d_j), i,j \in [1,n]$, and builds parallel MLPs for each group in order to produce intermediate group relevance scores. The final ranking score for a given document $d_k$ is calculated by accumulating its intermediate scores.

Other studies also extended the LambdaRank approach to general cases. LambdaLoss [Wang et al., 2018] provides a theoretical analysis of the effectiveness when directly optimizing an evaluation metric. It also generalizes LambdaRank to optimize other metrics such as Average Relevance Position (ARP) [Wang et al., 2018]. A toolkit for neural L2R [Pasumarthi et al., 2018] is also made available recently.

All the existing L2R approaches require to be provided with a set of features. In general, these features are extracted independently from ranking-function learning. It is possible that pre-trained features are not optimal for the ranker. It is more reasonable to learn features and ranking function simultaneously, so that they can influence each other. A deep neural model offers a flexible framework to implement such an integrated approach.

Based on the above observations, we propose an Integrated Learning Model (ILM) which incorporates the representation- and interaction-based features within the learning-to-rank framework. In this model, both the ranker and features are trained simultaneously in an end-to-end fashion by LambdaRank. We will describe the details of our model in the next section.

## 6.3. Integrated Learning Model

The general architecture of the Integrated Learning Model (ILM) is shown in Fig. 6.1. It integrates the learning of representation- and interaction-based features and the ranking function within a learning-to-rank framework. It is composed of the following components: representation module, interaction module, non-neural ranking features and the learning-to-rank layer.

### 6.3.1. Model Components

The proposed ILM is composed of several components. In the lower part, several modules aim at generating features. Those neural feature learning modules take query and document term embeddings as input which is obtained from an embedding look-up matrix shared by all neural feature learning modules.

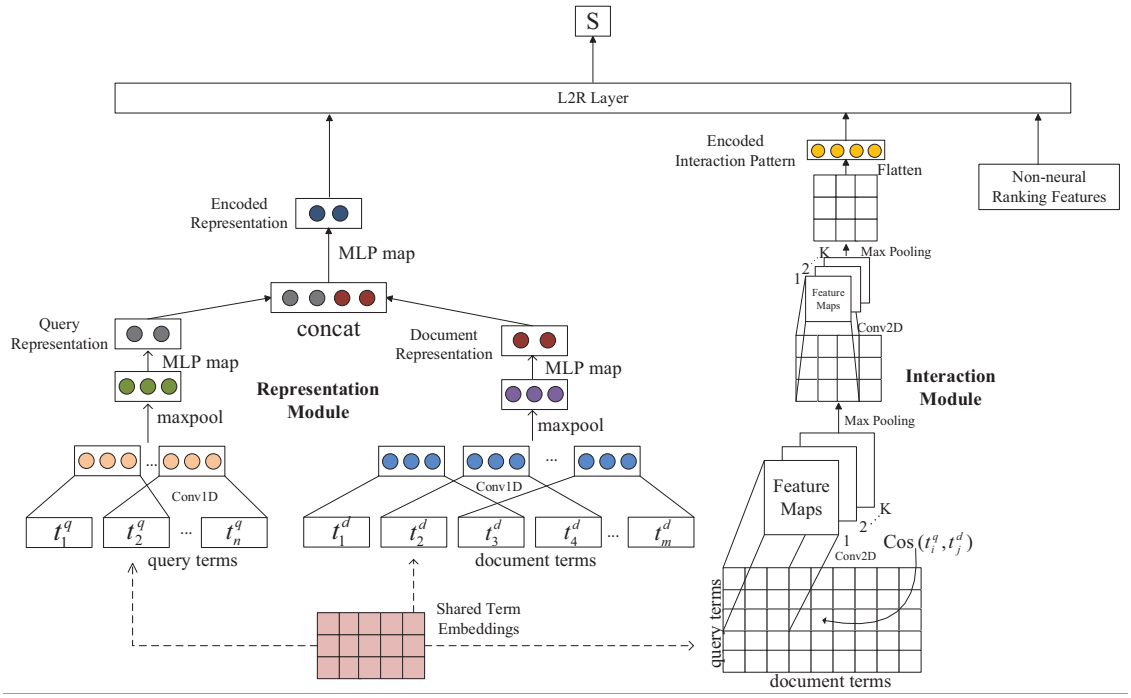



**Fig. 6.1.** Integrated Learning Model

We incorporate two types of neural modules to generate representation and interaction features. In addition to neural features, we also incorporate traditional (non-neural) features. The features will be fed into a L2R layer in order to produce a ranking score (S), and the neural features will be tuned together with the ranking function rather than being served as fixed inputs.

This framework is general and flexible. In fact, any existing representation-based and interaction-based neural models can be used to play the role of the two feature modules in the framework, provided that they can be trained in an end-to-end fashion together with the ranking function.

**Representation Module:** Inspired by the CDSSM [Shen et al., 2014] representation learning module, we employ a similar 1D convolutional model to learn query and document representations. The query and document are represented by a set of word embedding vectors $q = [t_1^{(q)},...,t_n^{(q)}]$, and $d = [t_1^{(d)},...,t_m^{(d)}]$, where $t_i^{(q)}$ and $t_j^{(d)}$ represent the embedding vectors for query term $i$ and document term $j$ respectively, and $n$, $m$ are the query length and document length respectively. Afterwards 1D convolution is applied to aggregate term embeddings inside a window of size $2k+1$ into phrase representations as follows.

$$C_{i,rep}^q = f(W^q * [t_{i-k}^q; ..; t_{i+k}^q] + b^q) \tag{6.3.1}$$

$$C_{i,rep}^d = f(W^d * [t_{i-k}^d; ..; t_{i+k}^d] + b^d) \tag{6.3.2}$$

where $C^q_{i,rep}$ and $C^d_{i,rep}$ represent the convolved representation for the $i^{th}$ query and document term respectively; $f$ is the activation function; $W$ and $b$ represent the weight and bias of the convolution; and $2k+1$ is the window size of the convolution.

Once the convolution is performed, a dimension-wise max-pooling is performed and the max-pooled query and document representations are fed into an output MLP layer for dimensionality reduction. The process is summarized as follows.

$$P^q_{rep} = \max_j(C^q_{i,rep}(j)) \quad P^d_{rep} = \max_j(C^d_{i,rep}(j)) \tag{6.3.3}$$

$$Q_{rep} = f(W^q_o P^q_{rep} + b^q_o) \quad D_{rep} = f(W^d_o P^d_{rep} + b^d_o) \tag{6.3.4}$$

where $P^q_{rep}$ and $P^d_{rep}$ are the max-pooled representations of the query and document; $C_{i,rep}(j)$ represents the $j^{th}$ dimension of the convolved vector $C_{i,rep}$; $Q_{rep}$ and $D_{rep}$ are the query and document representations after the dimension-reduction output layer.

The representation of query $Q_{rep}$ and document $D_{rep}$ are then concatenated and fed into a representation encoding MLP to produce the representation feature vector for the L2R layer. The process is defined as follows:

$$E_{rep} = concat(Q_{rep}, D_{rep}) \tag{6.3.5}$$

$$H_{rep} = f(W_{rep} E_{rep} + b_{rep}) \tag{6.3.6}$$

where $E_{rep}$ is the concatenation of $q_{rep}$ and $d_{rep}$; $f$ is the activation function; $W_{rep}$ and $b_{rep}$ are the weight and bias of the representation encoding MLP.

Notice that this last step represents an important difference with a standalone representation-based model: our module aims at producing a vector of neural representation about the document and the query, rather than a matching score. As we stated earlier, these features can interact in the L2R layer, together with the interaction features that we will describe in the following section.

**Interaction Module:** MatchPyramid is a popular interaction-based model which has shown promising performance in applications ranging from short text matching [Pang et al., 2016b] to IR [Pang et al., 2016a]. We build similar interaction module to learn interaction pattern from query-document pairs and employ the learned pattern as interaction feature.

First the local interaction matrix $I$ is built by applying cosine similarity between each query term embedding $t^{(q)}_i$ and document term embedding $t^{(d)}_j$.

$$I_{ij} = cos(t^{(q)}_i, t^{(d)}_j) \tag{6.3.7}$$

Once the input interaction matrix is constructed, a series of 2D convolution and max-pooling layers are added on top in order to build more abstract interaction patterns.

$$C^k_{1,inter} = f(W^k_1 * I + b^k_1),\ k = 1,..,K \tag{6.3.8}$$

$$P^k_{1,inter} = max\_pool(C^k_{1,inter}),\ k = 1,..,K \tag{6.3.9}$$

$$C^k_{i,inter} = f(W^k_i * P^k_{i-1,inter} + b^k_i),\ i = 2,..,L,\ k = 1,..,K \tag{6.3.10}$$

$$P^k_{i,inter} = max\_pool(C^k_{i,inter}),\ i = 2,..,L,\ k = 1,..,K \tag{6.3.11}$$

where $C^k_{i,inter}$ is the feature map $k$ of the $i^{th}$ convolved layer; $I$ is the input interaction matrix; $W^k_i$ and $b^k_i$ are the kernel and bias of layer $i$ for the feature map $k$; $L$ is the number of convolution layers, and $K$ is the number of feature maps; $f$ is a non-linear mapping; and $*$ represents the convolution operator.

In order to extract the interaction pattern of $q$ and $d$, following [Severyn and Moschitti, 2015, Pang et al., 2016a], the last max-pooled layer is flattened into a vector and fed into an interaction pattern encoding MLP. The encoded vector is then utilized as interaction feature and fed into the L2R layer.

$$E_{inter} = flatten(P_{L,inter}) \tag{6.3.12}$$

$$H_{inter} = h(W_o E_{inter} + b_o) \tag{6.3.13}$$

**Non-neural ranking features:** We concatenate scalar feature values together and reshape it into $BS \times N_{feats}$ matrix $H_{feats}$, where $BS$ is the batch size, $N_{feats}$ is the number of non neural features. These features are fed into the L2R layer. In this study, we employ the following non neural ranking features: BM25, LM scores between the query and document, term frequency of the document title, body, and whole document, IDF of document title, body and whole document, TF-IDF of document title, body and whole document, length of document title, body and whole document.

**L2R Layer:** The L2R layer aims at computing a ranking score $S$ based on the representation features $H_{rep}$, interaction feature $H_{inter}$ and non-neural ranking features $H_{feats}$:

$$S = f(W_s[H_{rep}; H_{inter}; H_{feats}] + b_s) \tag{6.3.14}$$

where $W_s$ and $b_s$ are the weight and bias and $f$ represents the activation function. Different from the Duet model which adds the ranking scores of the two matching models, we combine the neural features of the two matching mechanisms to encourage interactions between them. The ranking function will be trained through LambdaRank. In the same training process, the neural features will also be adapted.

### 6.3.2. LambdaRank End-to-End Training

We use LambdaRank model [Burges, 2010] and train the model in an end-to-end fashion. For a given query $q$, first, the probability that a document $d_i$ is more relevant than another document $d_j$ is modeled by the logistic function. Then, a cross entropy loss $C_{ij}$ is employed to measure the discrepancy between the ground truth label probability $\overline{P}_{ij} = \frac{1}{2}(1 + S_{ij})$ and the predicted probability $P_{ij}$, where $S_{ij}$ is the preference score of document $d_i$ and $d_j$ and takes values in $\{+1, 0, -1\}$.

$$P_{ij} = \frac{1}{1 + e^{-\sigma(s_i - s_j)}} \tag{6.3.15}$$

$$C_{ij} = -\overline{P}_{ij} log P_{ij} - (1 - \overline{P}_{ij}) log(1 - P_{ij}) \tag{6.3.16}$$

Afterwards the gradient $\frac{\partial C_{ij}}{\partial s_i}$ of loss function with respect to the predicted relevance score $s_i$ is multiplied by the change of nDCG [Wang et al., 2013] values by swapping the positions of $d_i$ and $d_j$ in the ranked list as follows.

$$\lambda_{ij} = \frac{\partial C_{ij}}{\partial s_i} |\Delta nDCG(i, j)| \tag{6.3.17}$$

The gradient of the loss $C_{ij}$ with respect to a trainable parameter $w_k$ of the model could be derived by the chain rule.

$$\frac{\partial C_{ij}}{\partial w_k} = \lambda_{ij} \frac{\partial (s_i - s_j)}{\partial w_k} \tag{6.3.18}$$

where the second factor $\frac{\partial (s_i - s_j)}{\partial w_k}$ is the difference of the gradients of the predicted scores with respect to the parameter $w_k$ which could be computed by back-propagation algorithm [Gu et al., 2017]. Note that for neural feature learning modules, we do not stop back-propagation at the L2R layer. We continue to back-propagate the loss signal to tune the neural feature learning layers and the word embeddings.

We train our model in a group-wise manner: for a given query $q$ and its corresponding documents $D = \{d_1, d_2, .., d_n\}$, we consider all possible pairs $(q, d_i, d_j)$ where $d_i, d_j \in D$. In a batch, there could be several query groups.

### 6.3.3. Alternative Configurations of ILM

To demonstrate the flexibility of ILM, we also test an alternative configuration, denoted as ILM-Hist, in which Deep Relevance Matching Model [Guo et al., 2016] (DRMM) is used as interaction feature module. The architecture is presented in Fig. 6.2

In the new interaction module, local interactions between query and document terms are mapped into histogram of $B$ bins. Akin to [Guo et al., 2016], for each query term $t_i^{(q)}$, we calculate the interaction of this query term with all document terms $[t_1^{(d)}, ..., t_m^{(d)}]$ by cosine similarity and count the number of interactions falling in each bin. We use log-based counts

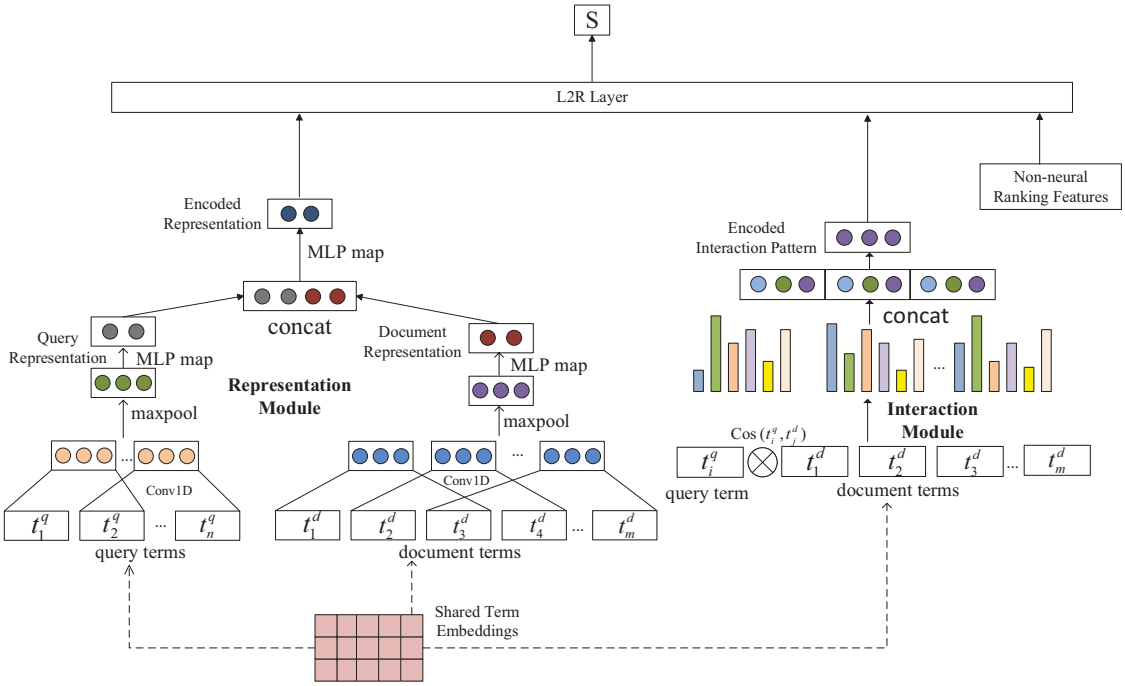



**Fig. 6.2.** Alternative Configuration ILM-Hist

as suggested by [Guo et al., 2016]:

$$I_i = [cos(t_i^{(q)}, t_1^{(d)}), ..., cos(t_i^{(q)}, t_n^{(d)})] \tag{6.3.19}$$

$$T_i = Hist(I_i, B) \tag{6.3.20}$$

$$T = concat([log(T_1), ..., log(T_m)]) \tag{6.3.21}$$

where $I_i$ is the interaction vector of query term $t_i^{(q)}$ with all document terms, $T_i$ is the histogram built based on the interaction vector $I_i$ for the $i^{th}$ query, and $T$ is the concatenated histograms of all query terms. The concatenated histograms $T$ are then mapped into the encoded interaction pattern $H_{hist}$ by a feed-forward layer.

$$H_{hist} = f(W_{hist}T + b_{hist}) \tag{6.3.22}$$

where $f$ is the activation function, $W_{hist}$ and $b_{hist}$ are the weight and bias of the model. This $H_{hist}$ is fed into the L2R layer as interaction features.

## 6.4. Experimental Setup and Datasets

In this section, we will present the collection used in this study, evaluation metrics and experimental settings. Those experiments aim to investigate the following research questions: Is integrated learning of features and ranking functions better than learning them separately? If so, how can we combine them effectively under the L2R framework? We will present the

collection and evaluation metrics used in this study and the experimental setups in the following subsections.

### 6.4.1. Collection

We employ the MQ2007 and MQ2008 datasets to conduct experiment for this study. The details of these datasets has been described in Section 3.1. We perform 5-fold cross validation and directly rank the validation/test fold rather than reranking. The relevance judgments are integers ranging from 0 to 2. For a given training query, we consider all $(q, d_i, d_j)$ pairs where the judgments are different. There are in average 12,886 and 2,799 training pairs over the 5 training folds of MQ2007 and MQ2008 respectively.

Since MQ2008 only contains 784 queries which is too small and will cause insufficient training problem in a deep model, we merge the training set of MQ2007 into that of MQ2008 and keep the validation and test set unchanged.

### 6.4.2. Baselines and Alternative Configurations

**Baselines:** We implement the **BM25** [Robertson and Zaragoza, 2009] and Language Model with Dirichlet smoothing (**LM**) with Indri[1] as our traditional model baselines. This comparison will help us to understand the performance of neural models with respect to those traditional models.

We also compare the performance of our models with RankMLP-Feats which employs Letor features, BM25 and LM scores. We feed them into the L2R layer trained with LambdaRank framework.

**Rep-MLP:** We first build models with only one matching mechanism. For representation-based model, we utilize the same representation-based module, denoted as Rep-MLP presented in Section 6.3.1 to learn representations.

**Inter-MLP:** For interaction-based model, denoted as Inter-MLP, we utilize the same pyramid-based interaction module presented in Section 6.3.1.

**HM-sum:** To combine the representation and interaction module without non-neural ranking features, following the mechanism of Duet Model [Mitra et al., 2017], HM-sum directly adds the matching scores of Rep-MLP and Inter-MLP to produce an aggregated relevance score.

**HM-MLP:** An alternative HM-MLP which employs the similar mechanism depicted in 6.3.1, which feeds the representation features and interaction features into an MLP to produce an aggregated relevance score, instead of directly adding the scalar representation and interaction matching score.

[1] https://www.lemurproject.org/indri/

All the above models are trained with the hinge loss.

$$L(Q,D_+,D_-;\Theta) = max(0, 1 - (S(Q, D_+) - S(Q,D_-))) \quad (6.4.1)$$

**ILM and its Variants:** We also build the models within our proposed ILM framework trained with LambdaRank in an end-to-end manner.
**ILM-Neu:** is the ILM model presented in Fig. 6.1 with only neural (representation and interaction) modules.
**ILM-BM25** and **ILM-LM** are the ILM-Neu model plus the baseline BM25 or LM score as non neural ranking feature.
**ILM-Feats** is the ILM model presented in Fig. 6.1 with both representation, interaction and additional Letor[1] features, BM25, LM scores as non neural ranking features.
**ILM-Fix-Feats:** To study the benefits of end-to-end learning of both the ranker and the neural feature learning modules, we also build model with pre-trained fixed neural features output by Rep-MLP and Inter-MLP. Those pre-trained fixed neural features are combined with Letor features, BM25 and LM scores.
**ILM-Hist:** To show that our framework is general and could be fit with other neural feature learning modules, we also build ILM-Hist which uses a histogram-based interaction module as presented in Section 6.3.3.

### 6.4.3. Evaluation Metrics

In this study, we employ Mean Average Precision (MAP), and Normalized Discounted Cumulative Gain (NDCGs) as evaluation metrics. They are presented in detail in Section 3.3. We calculate NDCG at cut-off position 1, 3, 10, 20. The MAP is a binary metric and aims to evaluate the general ad-hoc performance. The NDCGs at different cut-off positions simulate the real search scenario where a user would only want to browse the top-k ranked documents. We employ the official trec evaluation tools [2] to evaluate MAP and NDCGs.

Finally for statistical tests, we perform paired t-test with Bonferroni correction [Vialatte and Cichocki, 2008] on our ILM models with respect to baselines.

### 6.4.4. Experimental Setup

During indexing and retrieval, we process queries and documents with Krovetz stemmer [Krovetz, 2000] and remove stop words according to the Indri standard stop list[3]. We employ the pretrained 300-dimensional GloVe.6B.300d embeddings[4] to initialize the embedding look-up matrix and fine-tune it during training. For 1-layered and 2-layered representation

[1]https://www.microsoft.com/en-us/research/project/letor-learning-rank-information-retrieval/
[2]https://trec.nist.gov/trec_eval/; http://ir.cis.udel.edu/million/data.html#tools
[3]http://www.lemurproject.org/stopwords/stoplist.dft
[4]https://nlp.stanford.edu/projects/glove/

modules, following [Nie et al., 2018], we set the convolution window size to 3 and $[3, 5]$, the dimension of convolved vector to 256. For the 2-layered pyramid interaction module, we set the shape of the 2D convolution filters to be $(3, 3)$ and $(5, 5)$, the number of filters to be $[64, 32]$, and max-pooling shape to be $(2, 2)$, based on a preliminary study. For the histogram interaction module, we set the number of bins $B$ to 30 according to [Guo et al., 2016]. The dimension of the encoded representation $H_{rep}$ and interaction pattern $H_{inter}$ is set to 256, and the size of the L2R layer is set to 512. The vocabulary size is 400K. We set the max query length and document length to be $n = 15$, $m = 1000$, apply zero paddings [Pang et al., 2016a] and omit OOV document terms. We employ Adam optimizer to optimizer the trainable parameters of our models and the initial learning rate is set to $1 \times 10^{-3}$.

As for evaluation, since the MQ datasets have already been split into 5 folds by the author of the datasets, we perform 5 fold cross validation, using 3 folds for training, 1 fold for validation and the remaining fold for testing. Following the practice of [Fan et al., 2018, Pang et al., 2017], during evaluation, we do not rerank the validation and test instances over a baseline run, but directly rank the validation and test query-document pairs provided in the respective folds.

## 6.5. Experimental Results

In this section, we will present the experimental results of our proposed Integrated Learning Model and compare its performance with the baselines and variants.

**Table 6.1.** Experimental Results on MQ datasets [1]

| | MQ2007 | | | | | MQ2008 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Models | MAP | NDCG@1 | NDCG@3 | NDCG@10 | NDCG@20 | MAP | NDCG@1 | NDCG@3 | NDCG@10 | NDCG@20 |
| BM25 | 0.4584 | 0.3842 | 0.3872 | 0.4305 | 0.4849 | 0.4688 | 0.3524 | 0.4210 | 0.2183 | 0.1115 |
| LM | 0.4490 | 0.3630 | 0.3720 | 0.4145 | 0.4747 | 0.4569 | 0.3290 | 0.4073 | 0.2122 | 0.1086 |
| RankMLP-Feats | 0.4713 | 0.4241 | 0.4234 | 0.4535 | 0.5044 | 0.4789 | 0.4083 | 0.4486 | 0.2276 | 0.1206 |
| ILM-BM25 | $0.4918_{a}$ | $0.4376_{a}$ | $0.4398_{a}$ | $0.4745_{a}$ | $0.5248_{a}$ | $0.5069_{a}$ | $0.4132_{a}$ | $0.4715_{a}$ | $0.2375_{a}$ | $0.1240_{a}$ |
| ILM-LM | $0.4830_{b}$ | $0.4154_{b}$ | $0.4277_{b}$ | $0.4626_{b}$ | $0.5161_{b}$ | 0.5022 | $0.4170_{b}$ | $0.4583_{b}$ | $0.2367_{b}$ | $0.1224_{b}$ |
| ILM-Feats | $\mathbf{0.4987}^{c}_{*}$ | $\mathbf{0.4656}^{c}_{*}$ | $\mathbf{0.4574}^{c}_{*}$ | $\mathbf{0.4874}^{c}_{*}$ | $\mathbf{0.5342}^{c}_{*}$ | $\mathbf{0.5160}^{c}_{*}$ | **0.4315** | $\mathbf{0.4731}^{c}_{*}$ | $\mathbf{0.2450}^{c}_{*}$ | $\mathbf{0.1278}^{c}_{*}$ |

The main experimental results are presented in Table 6.1. We conduct paired t-test to compare ILM-BM25, ILM-LM, ILM-Feats with their respective counterparts BM25, LM, RankMLP-Feats respectively. The statistically significant results ($p < 0.05$) with respect to BM25, LM and RankMLP-Letor are marked with $a, b, c$ respectively. For our proposed model ILM-Feats, we also perform Bonferroni correction with respect to the group of all

[1] Statistical significance (p<0.05) with respect to BM25, LM and RankMLP-Feats is marked with $a$, $b$ and $c$. $^{*}$ indicates statistical significance (p<0.05) with Bonferroni correction with respect to the 3 baselines.

**Table 6.2.** Comparison of Integrated and Separate Learning of the Ranker and Features [2]

| | MQ2007 | | | | | MQ2008 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Models | MAP | NDCG@1 | NDCG@3 | NDCG@10 | NDCG@20 | MAP | NDCG@1 | NDCG@3 | NDCG@10 | NDCG@20 |
| ILM-Fix-Feats | 0.4694 | 0.4041 | 0.4152 | 0.4507 | 0.5003 | 0.4881 | 0.4000 | 0.4486 | 0.2368 | 0.1243 |
| ILM-Feats | $\mathbf{0.4987}_d$ | $\mathbf{0.4656}_d$ | $\mathbf{0.4574}_d$ | $\mathbf{0.4874}_d$ | $\mathbf{0.5342}_d$ | $\mathbf{0.5160}_d$ | **0.4315** | $\mathbf{0.4731}_d$ | $\mathbf{0.2450}_d$ | $\mathbf{0.1278}_d$ |

3 baselines, and the statistically significant results after Bonferroni correction are marked with $*$. From Table 6.1, we can first observe that our proposed ILM-Feats outperforms all 3 baselines on all evaluation measures. This confirms the effectiveness of our proposed ILM model. To further understand the advantage of our proposed ILM model, we examine the following questions in the experiments:

**(1) Are neural features useful?**

To answer the question more specifically, we can compare models with neural features (ILM-BM25, ILM-LM, ILM-Feats) against their counterparts that do not contain neural features (BM25, LM, RankMLP-Feats) in Table 6.1. In all the cases, we observe a large improvement on all the evaluation measures, and the differences are statistically significant. This result clearly demonstrates that the neural features on representation and interaction are useful, and they can help improve the effectiveness even when a set of traditional features are already included.

**(2) Is integrated learning better than separate learning of the ranker and features?**

To investigate the benefits of integrated learning of the ranker and neural features, we build ILM-Fix-Feats. In this models, we learn representation and interaction neural features with separate models, then input them as fixed features to the L2R layer and perform training by LambdaRank. Comparing it with the corresponding end-to-end version (ILM-Feats), from Table 6.2, we observe that the integrated learning is better than separate learning. This result confirms the advantage we expected with integrated learning. Although it is difficult to visualize the interactions between feature learning and ranking-function learning, we believe that the mutual influence between them reinforced both learning processes and this contributed to obtaining both better features and a better ranking function.

## 6.6. Discussions and Analysis

In this section, in order to better understand the advantage of integrated learning of features and ranking function, we will discuss the experiment results, present some case studies based on some typical queries in the test collection and give the complexity analysis of our proposed Integrated Learning Model.

[2]Statistical significance ($p<0.05$) with respect to the ILM-Fix-Feats is marked with $d$

### 6.6.1. Discussions

**Table 6.3.** Experimental Results of Alternative Configurations [1]

| | MQ2007 | | | | | MQ2008 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Models | MAP | NDCG@1 | NDCG@3 | NDCG@10 | NDCG@20 | MAP | NDCG@1 | NDCG@3 | NDCG@10 | NDCG@20 |
| Rep-MLP | 0.4199$_*$ | 0.3169$_*$ | 0.3364$_*$ | 0.3828$_*$ | 0.4450$_*$ | 0.4085$_*$ | 0.2738 | 0.3391$_*$ | 0.1837$_*$ | 0.0931 |
| Inter-MLP | 0.4156$_*$ | 0.3129$_*$ | 0.3298$_*$ | 0.3832$_*$ | 0.4443$_*$ | 0.4115$_*$ | 0.2797 | 0.3479$_*$ | 0.1864$_*$ | 0.0922$_*$ |
| HM-Sum | 0.4169$_*$ | 0.3087$_*$ | 0.3301$_*$ | 0.3839$_*$ | 0.4452$_*$ | 0.4222$_*$ | 0.3078 | 0.3599$_*$ | 0.1940$_*$ | 0.0961$_*$ |
| HM-MLP | 0.4245$_*$ | 0.3301$_*$ | 0.3446$_*$ | 0.3938$_*$ | 0.4505$_*$ | 0.4280 | 0.3116 | 0.3762$_*$ | 0.2007$_*$ | 0.1005$_*$ |
| ILM-Neu | 0.4313 | 0.3494 | 0.3586$_*$ | 0.3986$_*$ | 0.4589$_*$ | 0.4593 | 0.3639 | 0.4128$_*$ | 0.2124$_*$ | 0.1053$_*$ |
| ILM-Hist | **0.4884**$_*$ | **0.4487**$_*$ | **0.4458**$_*$ | **0.4739**$_*$ | **0.5224**$_*$ | **0.5192**$_*$ | **0.4345**$_*$ | **0.4835**$_*$ | **0.2478** | **0.1295**$_*$ |

**Focus of representations and interactions:** As discussed earlier, the focus of matching in representation- and interaction-based models is different. Representation-based models focus on representing contents, while interaction-based models focus more on local lexical matches.

To confirm the roles of representation- and interaction-based models, we extract some typical conceptual and lexical queries from Million Query and compare the performance of Rep-MLP, Inter-MLP and HM-MLP on NDCG@10 in Table 4.1. From Table 4.1, we

**Table 6.4.** NDCG@10 of Representative Queries

| topic_num | Query | Rep-MLP | Inter-MLP | HM-MLP |
|---|---|---|---|---|
| 9394 | preventing alcoholism | 0.75297 | 0.61683 | **0.85124** |
| 9963 | small business statistics | 0.11068 | 0.04793 | **0.40252** |
| 8023 | voyager 2 Neptune | 0.4807 | 0.65641 | **0.73464** |
| 8068 | distance between grand canyon and phoenix | 0.15508 | 0.20615 | **0.22341** |

can observe that for conceptual queries (top part) such as "preventing alcoholism", "small business statistics", etc., the representation-based model Rep-MLP outperforms interaction-based model Inter-MLP. Actually all those queries expect some degree of generalization or expansion from the term space. For example, for the query "preventing alcoholism", documents containing "stop alcohol abuse" are also relevant and for the query "small business statistics", documents containing "small company statistics" are also relevant.

For lexical queries (bottom part) about some people and places, such as "voyager 2 Neptune", "distance between grand canyon and phoenix" which require exact term match, the interaction-based model Inter-MLP outperforms the representation-based model Rep-MLP. Take the query "distance between grand canyon and phoenix" as example, the user

[1] $_*$ indicates statistical significance ($p<0.05$) with Bonferroni correction with respect to the baselines in Table 6.1.

issued this query would like to find the distance between grand canyon and phoenix, but not other places, therefore interaction-based models which focus more on lexical match performs better.

By combing the representations and interactions, the model HM-MLP outperforms both Rep-MLP and Inter-MLP for both lexical and conceptual queries. This result demonstrates the complementarity between the two types of features. The general performance of the three models are presented in Table 6.3. We can observe that the model using only one type of feature (Rep-MLP vs Inter-MLP) can yield equivalent performance. When both types of features are combined in HM-MLP, we obtain better results. This confirms again the two types of features are complementary.

**Integrated vs Separated Learning of Features and Ranking Function:** In section 6.5, we have already observed that integrated learning of the neural features and ranking function (ILM-Feats) is better than learning them separately (ILM-Fix-Feats). To better illustrate the strength of integrated learning of features and ranking function, we take a representative query "qid 7993: model railroads" from the test set and print the top 10 documents ranked by ILM-Fix-Feats and ILM-Feats in Fig. 6.3. Note that those two models share the same neural components and non neural ranking features, but differ in whether the neural features are learned together with the L2R ranking layer. ILM-Fix-Feats employs pre-trained fixed representation and interaction features whereas ILM-Feats has the neural features trained in an end-to-end manner with the ranker.

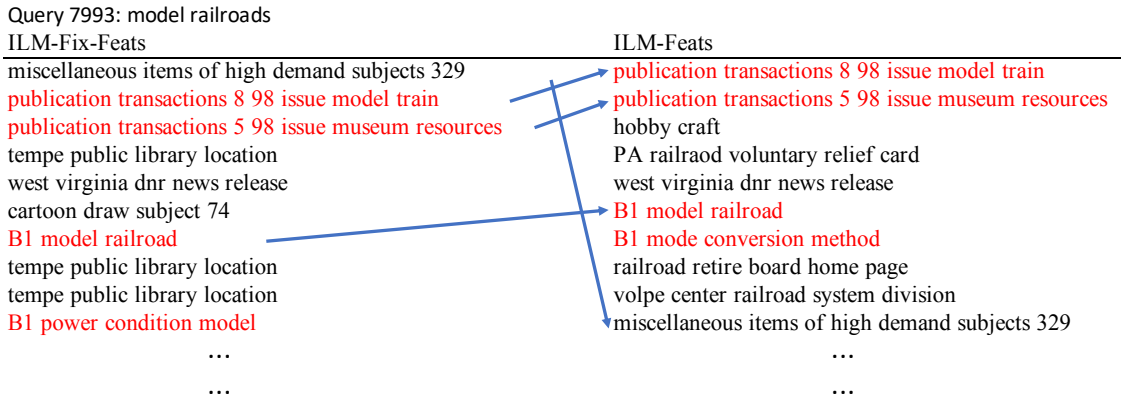



**Fig. 6.3.** Rank List for Query 7993: Document titles of relevant documents are marked in red, and non relevant ones are marked in black; ILM-Fix-Feats represents the model which learns neural features and ranking function separately, ILM-Feats learns the neural features and ranking function in an end-to-end manner

The document titles of relevant documents (judgments $\geq 1$) are marked in red, and non relevant documents (judgments $= 0$) are marked in black. By comparing the 2 ranking lists, we can observe that relevant documents are pushed upwards and non relevant documents are pushed downwards in the rank list produced by the model trained in an end-to-end manner.

For example, the document entitled "publication transactions 8 98 issue model train" which is relevant to this query is pushed upwards from rank position 2 to position 1 in the end-to-end ILM model. Another document entitled "B1 model railroad" which is relevant is pushed up from position 7 to position 6. Finally, the document entitled "miscellaneous items of high demand subjects 329" which doesn't talk about railroad models is non relevant, and it is pushed down from position 1 to position 2 in the end-to-end ILM model. This shows that within the L2R framework, if we integrate the learning of the neural feature modules and the ranker in an end-to-end manner, relevant documents could be ranked further upwards, resulting in improved performance.

We also take another representative query "qid 9150: sarcoma" (a type of tumor) from the test set and print the top 10 documents ranked by ILM-Fix-Feats and ILM-Feats in Fig. 6.4. Same as the above setting, the two models share the same neural components and non neural features but only differ in whether the neural representation and interaction features are learned in an end-to-end manner or not. Same as the above example, the document

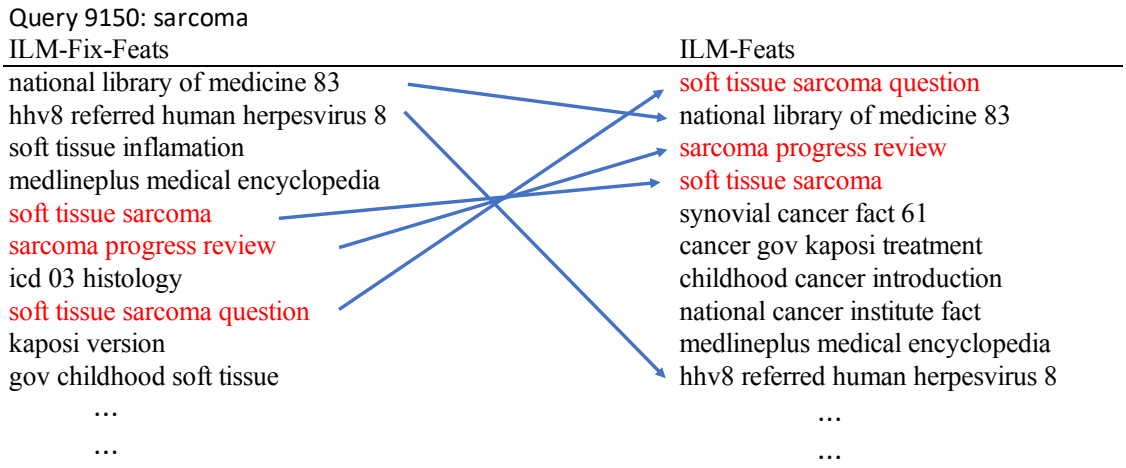



**Fig. 6.4.** Rank List for Query 9150: Document titles of relevant documents are marked in red, and non relevant ones are marked in black; ILM-Fix-Feats represents the model which learns neural features and ranking function separately, ILM-Feats learns the neural features and ranking function in an end-to-end manner

titles of relevant documents (judgments $\geq 1$) are marked in red, and non relevant documents (judgments $= 0$) are marked in black. By comparing the 2 ranking lists, we observe the same phenomenon: Relevant documents are pushed upwards and non relevant documents are pushed downwards in the rank list produced by the model trained in an end-to-end manner. In particular, the relevant document entitled "soft tissue sarcoma" is pushed up from position 5 to position 4, the relevant document entitled "sarcoma progress review" is pushed up from position 6 to position 3, and the relevant document entitled "soft tissue sarcoma question" is pushed up from position 8 to position 1. On the other hand, the non relevant documents "national library of medicine 83" and "hhv8 referred human herpesvirus

8" are pushed down from position 1 and 2 to position 2 and 10. This example illustrates again that within the L2R framework, if we integrate the learning of the neural feature modules and the ranker in an end-to-end manner, relevant documents could be ranked further upwards, resulting in improved performance.

**Effectiveness of LambdRank vs. Hinge Loss:** ILM-Neu and HM-MLP are two similar models that share the same model architecture and use the same neural features. However, the former uses LambdaRank framework to optimize while the latter uses hinge loss. From Table 6.3, we can observe that ILM-Neu outperforms HM-MLP on all evaluation metrics on both datasets. This comparison demonstrates the benefits of employing LambdaRank framework over the traditional hinge loss training approach and confirms the result in [Wang et al., 2018]. In fact in the Lambdarank framework we directly optimize the delta NDCG which has the effect of pushing the relevant documents upwards and pushing the non relevant documents downwards. However, the traditional hinge loss doesn't have this group-wise global view and could only optimize locally during training, therefore resulting in worse results than the model trained with Lambdarank framework.

**Combining features vs. combining scores:** One of our initial intuitions is that it is better to combine representation and interaction features than combining the scores they produce. This can be confirmed by comparing HM-MLP and HM-sum (equivalent to the Duet model) in Table 6.3: The HM-MLP employs a MLP to combine the representation features, whereas the HM-sum simply sums the scalar scores of the representation and interaction modules. From this comparison, It is clear that combining features is better than combining scores. This suggests that MLP can indeed make better use of the features when they are presented together since this allows them to have possible interactions among the latent dimensions, whereas the simple addition of the two scalar scores would possibly omit useful signals which may be helpful for the model.

**Flexibility of ILM:** Our ILM is also general, it is possible to replace the representation/interaction module with different ones. To illustrate this possibility, for the interaction module, we replace the pyramid-based one with the histogram-based one depicted in Fig. 6.2 and build ILM-Hist. Experimental results in Table 6.3 show that it could still outperform traditional baselines and offer comparable performance with respect to the original ILM. This demonstrates the flexibility of our proposed ILM model: So long as there are neural representation or interaction modules which are fine-tunable together with the L2R layer, it could be plugged into our proposed ILM model. Moreover, if additional non-neural models are deemed beneficial, they could also be easily incorporated into our ILM model at the L2R layer.

### 6.6.2. Complexity Analysis

In this subsection, we analyze the time complexity of our proposed Cross-level Matching Model. Our representation modules are based on either 1D-CNN. According to [Vaswani et al., 2017], one 1D convolution layer will cost $O(k * n * d)$ multiplications in inference/test mode or equal amount of gradient operations in training mode, where $k$ is the dimension of the convolved layer vectors, $d$ is the dimension of the input layer vectors, and $n$ is the sequence length (number of terms).

Our interaction modules are built with 2D convolutions. According to [He and Sun, 2015], one 2D convolution layer will cost $O(n_{in} * s_f^2 * n_{out} * m_h * m_w)$ operations, where $n_{in}$ is the number of input channels, $s_f$ is the length of filters (in our setting it's squared shape, so is the $s_f^2$), $n_{out}$ is the number of output features maps, and $m_h$, $m_w$ are the height and width of output feature map.

For our Integrated Learning Model in Fig. 6.1, suppose that the maximum query and document length are $n$ and $m$ respectively, the input embedding dimension is $d$, the hidden size is $h$, and in our setting $h < d$. Suppose we feed in a batch of $B$ triples $(q, d_i, d_j)$. The representation module would cost $B * (n + m) * h * d$ operations.

As for the interaction module, suppose that there will be no more than $k$ feature maps and the max filter sizes are $s_f * s_f$, then for each 2D convolutional layer, it will cost at most $B * k^2 * s_f^2 * n * m$ operations. The construction of the interaction matrix will cost at most $B*n*m$ operations, and there would be an extra $B*(n*m*h+h*1)$ operations for the final MLP layer. Therefore, for the interaction module there would be $2B*(k^2*s_f^2+h+1)*n*m+2h$ operations.

The concatenation of non neural ranking features will cost a constant operations O(1), and the learning to rank layer involves calculating NDCG of the current batch which will cost $O(\alpha * Blog(B))$, where the $Blog(B)$ reflects the sorting involved in calculating ideal DCG.

Therefore the whole model will have a complexity of $O(B * [(n + m) * h * d + 2(k^2 * s_f^2 + h + 1) * n * m + 2h + \alpha log(B)])$. if the hyper-parameters of the network are fixed (i.e. $d$, $s_f$, $k$, $h$, $B$ are fixed), then the complexity of the model is dominated by $n * m$ which is bi-linear to the length of query $n$ and document $m$.

## 6.7. Conclusion

Recent neural IR models have demonstrated their potentials. Those models mainly employ either representation or interaction mechanism or a combination of the two. Those neural models are good at feature learning, however they usually employ simple ranking function which is another crucial component of an IR model.

Learning-to-rank (L2R) framework provides a good solution to learn a better ranking function. However we still observe a large gap between traditional L2R approaches and neural models, since traditional L2R usually accept fixed, hand-crafted features as input. Therefore it is natural to investigate whether it is possible to take advantage of both the feature learning power of neural models and the good ranking function learned by L2R by combining the neural models of different mechanisms under the L2R framework.

To investigate this question, In this chapter, we proposed an Integrated Learning Model (ILM) to combine both representation-based, and interaction-based and non neural features within the LambdaRank framework. Both the neural feature modules and the ranking function are trained in an end-to-end fashion to allow interactions between them. Experiments on public datasets confirm the effectiveness of integrated learning of ranking features of different nature and ranking function.

# Chapter 7

# Conclusion and Future Work

In this section we will give a general conclusion by summarizing the work presented in this thesis and point out possible future research directions.

## 7.1. Summary of the Thesis

In this thesis, we present 4 investigations focusing on better adapting neural networks for Information Retrieval tasks. The first two works focus on learning multiple layers of representations or interactions and estimating the global relevance score by considering the contribution of matching of all layers so as to satisfy different types of query (lexical and conceptual). The third work focuses on matching representations of different layers in order to achieve term-phrase matching. Finally the forth work focuses on learning neural features and ranking function simultaneously in an end-to-end manner in order to take advantage of both the feature-learning power of neural models and the better ranking function learned by learning-to-rank framework.

When the first two investigations are made, typical neural models for IR are either based on representations or on interactions. We observe that most of the existing neural IR models (regardless of their mechanism) only employs the last layer representation/interaction to produce a matching score as global relevance score. This might not be an optimal approach for neural IR models, since users query could be of different types: some queries might be lexical query where an exact match on the term level is more appropriate, whereas others might be conceptual query which requires a certain degree of generalization to match with relevant documents, and there could also be queries in between. Since neural models could learn several layers of representations/interactions which generalize terms into more and more abstract semantics, it is worthwhile to take advantage of the matching signals from multiple layers to satisfy different types of queries rather than only using the matching information of the last layer.

To achieve this goal, in the first two articles, we proposed Multi-level Abstraction Convolution Model (MACM) which produces a matching score at every layer and aggregates them by a dynamic gating mechanism. Therefore this MACM is capable of coping with different types of queries by matching them with documents at different levels of abstraction. Our contribution is two-fold: (1) We design a multi-level matching mechanism which leverages the matching signals of all layers between query and document to satisfy different types of query. (2) We conduct experiment on public dataset which validates the advantage of multi-level matching over single-level matching.

Next, we explored the problem of term-phrase matching. As the writing of the third article, many neural IR models have been proposed and shown competitive results. In particular, interaction-based models have shown superior performance to traditional models in a number of studies. However, the interactions used as the basic matching signals are between single terms or their embeddings. For instance, in MatchPyramid, the basic matching components are term embeddings, and this model employs 2D convolutions to analyze the term-to-term interaction matrix. In DRMM, the basic matching components are also term embeddings. This model first calculates cosine similarity between each query term embedding with all document term embeddings and then convert the similarities into histogram.

In reality, a term can often match a phrase or even longer segment of text. Sometimes the longer segment of text doesn't even contain exact matched term, but the whole segment could represent a certain concept which could be matched with a single term. In theses cases, apart from using term-to-term local interactions, it is also helpful to add additional matching signals between term and phrase to help the model to cope with the need of term-phrase matching.

To achieve this, in the third work, we proposed a Cross-level Matching Model (CLMM). We first employ 1D-CNN or BiLSTM to learn representations of the phrases. Afterwards, in addition to the traditional term-to-term matching channel in MatchPyramid, we also match term representation with phrase representation between query and document. Those additional matching signals and the traditional term-to-term matching signals are combined to produce a final relevance score through a gating mechanism to pay more attention to more important channels. Our contribution is two-fold: (1) We propose a Cross-level Matching Model to consider not only term-to-term matchings but also term-phrase matching signals to satisfy the need of term-phrase matching. (2) We conduct experiments on public datasets and validate our hypothesis that incorporating term-phrase matching alongside with term-to-term matching signals will help to improve the performance of an neural IR model.

Finally, we investigated the integration of neural feature learning and ranking function learning. Any learning-based IR model boils down to essentially two parts: features and ranking function. As the writing of the last article, a lot of neural IR models have demonstrated their powerful feature learning capability thanks to the neural feature learning modules they

employed, yet most of them simply output a scalar relevance score and employed a simple pairwise loss to train.

When it comes to learning a better ranking function, the learning-to-rank framework (L2R) such as LambdaRank provided a effective solution. However, we still observe a gap between neural IR models and the L2R framework: the traditional L2R framework only accept fixed features which are often hand-crated and lacks the flexibility and expressive power exhibited in neural features.

Therefore it is worthwhile to investigate whether it is possible to take advantage of both the feature-learning power of neural models and the good ranking function learned by L2R framework by integrating them together. In order to answer this question, in the last article, we proposed an Integrated Learning Model which integrates the learning of both neural features and L2R ranking function. Our contribution is three-fold: (1) We combine representation-based and interaction-based neural approaches in a flexible learning-to-rank framework. (2) We integrate feature learning with L2R ranking function learning, which are trained end-to-end simultaneously. (3) We show that the proposed model can outperform the existing neural models on public datasets, and that integrating traditional features can further improve the performance.

## 7.2. Future Work

In this thesis, after observing the limitations in existing neural IR models, we provide some solutions to better adapt neural models to IR tasks. Recently, with the recent development of NLP tools and more sophisticated deep learning techniques, it is possible to further improve the performance of neural IR models. In this subsection, we will discuss some possible future research directions following this thesis.

Recently the rapid development of pre-trained language models [Radford et al., 2019, Devlin et al., 2019, Yang et al., 2019, Lewis et al., 2020] have shown their potentials in many natural language processing tasks, ranging from traditional tasks such as masked token prediction to natural language understanding tasks such as hate speech recognition [Sohn and Lee, 2019].

Specifically, the BERT model [Devlin et al., 2019] whose architecture is based on the self-attention mechanism [Vaswani et al., 2017] has shown its effectiveness and versatility in many applications. One of the advantage of the BERT pre-trained model is that it contains multiple layers of very detailed and expressive representations learned through some large training corpus (800M and 2500M tokens) and could be adapted to perform different downstream tasks by fine-tuning it with the task specific module or objective function. In fact, [van Aken et al., 2019] has analyzed the roles of different BERT layers in the question answering (QA) tasks. They found out that the lower layers are responsible for semantic clustering,

the middle layers help to connect entities to mentions and attributes, higher layers are able to match questions to supporting facts in documents, and finally the last layer is responsible for answer extraction. These findings are in line with the property of neural models where at higher layers, more abstract representations/semantics are constructed to perform more abstract tasks.

Thanks to the good representations learned by BERT, there have been some studies using pre-trained BERT and fine-tuning it for IR tasks. For example, [Dai and Callan, 2019] showed that BERT could help better understand the query and document texts with the help of contextual terms. MarkedBERT [Boualili et al., 2020] adds marker tokens around query exact-matched terms and those appearing in documents in order to introduce explicitly exact match signals to further improve performance.

Recently the Dense Retrieval [Zhan et al., 2020] framework has also attracted much attention thanks to the use of the pre-trained language models (PLMs) like BERT. In those dense retrieval models, instead of concatenating query and document together to learn a query-document representation like the work presented above, query and document representations are constructed separately by a Siamese structure composed of two PLMs. Afterwards the query and document representation are matched through a fast matching function such as dot product or cosine similarity. This framework presents two huge advantages for both research and industrial applications: (1) The costly representation learning process for documents could be moved to indexing stage to construct dense representations for each document in the collection, therefore providing much faster query serving responses [Hofstätter et al., 2020]. When a user issues a query, the dense retrieval model will build dense query representation very quickly (since queries are usually short) and match it with pre-built dense document representations by the fast matching function. (2) The neural model could be employed to perform full collection search instead of reranking a pre-fetched set of candidates by a traditional retrieval method such as BM25. In fact, when user issues a query, the model will construct the dense query representation and perform an approximate nearest neighbor search [Babenko and Lempitsky, 2016] through the pre-built document dense index [Xiong et al., 2021].

For example, in TwinBERT [Lu et al., 2020a], the query and document representations are learned by two symmetrical BERTs and the last layer's [CLS] representations of query and document are employed as global representations to perform a match by dot product. The ColBERT [Khattab and Zaharia, 2020] model also learns query and document representations separately by two BERT encoders. Afterwards, each of the last layer's query token representation is matched with the last layer's document token representations, and a max interaction value is retained. Finally the max interactions are aggregated for all query terms. By delaying the interactions until the completion of query/document representation construction, it could also achieve the desired fast response for serving queries.

Although those dense retrieval models have demonstrated their effectiveness, there are still circumstances where they could not outperform a traditional sparse retrieval model [Lin et al., 2020, Qiao et al., 2019, Luan et al., 2021], especially when the documents are long or it requires exact match between query and document. This implies that in order to fully harness the power of BERT-based dense retrieval models for IR tasks, some adaptations need to be made.

One of the possible adaptations that could be made is to take advantage of multiple layers learned by BERT encoders. In fact, most existing BERT-based dense retrieval models only take advantage of the last BERT layer to output a global matching score. However as we proposed in this thesis, using multiple layers to participate in ranking could make the model suitable for different types of query, thereby improving performance. As presented above, since the pre-trained layers in BERT naturally present the property of hierarchical abstractions [van Aken et al., 2019], it is possible to adapt BERT to follow the multi-level matching mechanism proposed in this thesis. Since the dense representations of all BERT layers are pre-built during indexing stage, if we adopt the same fast matching function such as dot product, the response time for serving a query only scales to a factor of numbers of layers participated in ranking, therefore still suitable for the need of fast retrieval.

Another challenge to adapt BERT-based dense retrieval models for IR tasks is to handle the long document. BERT was designed to handle sentence to passage level text which ranges from tens to a few hundreds terms. However, in the application of IR, documents could be as long as thousands of terms. In this situation, feeding the whole document into the model might make the model lose focus and give poor results. To address this issue, one possible solution is to cut the document into passages and build the query and passage representations by BERT-encoder during indexing time while maintaining a link from each passage to the original document it belongs to. Afterwards one could perform a fast MaxSim operation as in [Khattab and Zaharia, 2020] to identify the best matched passages and their original document. A final relevance score between the document and the query could be obtained by aggregation of those passage level scores. Another solution is to base a retrieval model on the Longformer [Beltagy et al., 2020] architecture which performs dilated sliding window attention to save memory, therefore appropriate to encode very long documents.

Finally the basic element in existing BERT-based dense retrieval models are term representations, and it only implicitly considers the term dependencies through the self-attention mechanism. However in IR applications, there's sometimes the need to perform phrase level match. For example if the query contains the phrase "space exploration" and this phrase also appears in the document, then this constitutes a strong evidence that this document is relevant to the query. To explicitly model the phrase matching signals in BERT, one solution is to first identify the phrases in query and add the same marker to all terms within

this phrase, and perform the same marking operation in the document. This will allow the BERT model to automatically identify the existence of query phrases in document.

Overall we believe that information retrieval could benefit greatly from the recent advancements and applications of neural learning techniques, yet traditional features which are robust yet cheap to obtain could also play an role to assist a neural ranking model. However, direct applying those techniques to IR tasks may not be optimal since it neglects the characteristics of IR task itself. Therefore there are always some beneficial adaptations that one could make to better take advantage of those neural techniques in IR tasks. In this thesis, we discussed some of them and we hope that these mechanisms could spark more ideas and give some insights for researchers in this domain in the future.

# References


Eneko Agirre, Enrique Alfonseca, Keith B. Hall, Jana Kravalova, Marius Pasca, and Aitor Soroa. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, May 31 - June 5, 2009, Boulder, Colorado, USA*, pages 19–27. The Association for Computational Linguistics, 2009.

Qingyao Ai, Xuanhui Wang, Nadav Golbandi, Michael Bendersky, and Marc Najork. Learning groupwise scoring functions using deep neural networks. *CoRR*, abs/1811.04415, 2018.

Akiko N. Aizawa. An information-theoretic perspective of tf-idf measures. *Inf. Process. Manag.*, 39(1):45–65, 2003.

Hirotogu Akaike. Information theory and an extension of the maximum likelihood principle. In *Selected Papers of Hirotugu Akaike*, pages 199–213. Springer, 1998.

Avinash Atreya and Charles Elkan. Latent semantic indexing (LSI) fails for TREC collections. *SIGKDD Explor.*, 12(2):5–10, 2010.

Artem Babenko and Victor S. Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2055–2063. IEEE Computer Society, 2016.

Bing Bai, Jason Weston, David Grangier, Ronan Collobert, Kunihiko Sadamasa, Yanjun Qi, Olivier Chapelle, and Kilian Q. Weinberger. Learning to rank with (a lot of) word features. *Inf. Retr.*, 13(3):291–314, 2010.

Nicholas J. Belkin, Paul B. Kantor, Edward A. Fox, and Joseph A. Shaw. Combining the evidence of multiple query representations for information retrieval. *Inf. Process. Manag.*, 31(3):431–448, 1995.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020. URL `https://arxiv.org/abs/2004.05150`.

Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012. URL `http://arxiv.org/abs/1206.5538`.

Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, 2013.

Daniel Bikel and Imed Zitouni. *Multilingual natural language processing applications: from theory to practice*. IBM Press, 2012.

Holger Billhardt, Daniel Borrajo, and Victor Maojo. A context vector model for information retrieval. *J. Assoc. Inf. Sci. Technol.*, 53(3):236–249, 2002.

Cedric De Boom, Steven Van Canneyt, Thomas Demeester, and Bart Dhoedt. Representation learning for very short texts using weighted word embedding aggregation. *Pattern Recognit. Lett.*, 80:150–156, 2016.

Lila Boualili, Jose G. Moreno, and Mohand Boughanem. Markedbert: Integrating traditional IR cues in pre-trained language models for passage retrieval. In Jimmy Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu, editors, *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 1977–1980. ACM, 2020.

Siddhartha Brahma. Suffix bidirectional long short-term memory. *CoRR*, abs/1805.07340, 2018. URL `http://arxiv.org/abs/1805.07340`.

Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. Learning to rank using gradient descent. In Luc De Raedt and Stefan Wrobel, editors, *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pages 89–96. ACM, 2005.

Christopher J. C. Burges, Robert Ragno, and Quoc Viet Le. Learning to rank with non-smooth cost functions. In Bernhard Schölkopf, John C. Platt, and Thomas Hofmann, editors, *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 193–200. MIT Press, 2006.

Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In Zoubin Ghahramani, editor, *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*, pages 129–136. ACM, 2007.

Pablo Castells, Miriam Fernández, and David Vallet. An adaptation of the vector-space model for ontology-based information retrieval. *IEEE Trans. Knowl. Data Eng.*, 19(2): 261–272, 2007.

Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In Olivier Chapelle, Yi Chang, and Tie-Yan Liu, editors, *Proceedings of the Yahoo! Learning to Rank Challenge, held at ICML 2010, Haifa, Israel, June 25, 2010*, volume 14 of *JMLR Proceedings*, pages 1–24. JMLR.org, 2011.

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In Alexandros Karatzoglou, Balázs Hidasi, Domonkos Tikk, Oren Sar Shalom, Haggai Roitman, Bracha Shapira, and Lior Rokach, editors, *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, DLRS@RecSys 2016, Boston, MA, USA, September 15, 2016*, pages 7–10. ACM, 2016.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734, 2014.

Stéphane Clinchant and Éric Gaussier. Information-based models for ad hoc IR. In Fabio Crestani, Stéphane Marchand-Maillet, Hsin-Hsi Chen, Efthimis N. Efthimiadis, and Jacques Savoy, editors, *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010*, pages 234–241. ACM, 2010.

Stéphane Clinchant and Éric Gaussier. Probabilistic models for information retrieval. In Éric Gaussier and François Yvon, editors, *Textual Information Access: Statistical Models*, pages 3–32. Wiley-ISTE, 2012.

David Cossock and Tong Zhang. Subset ranking using regression. In Gábor Lugosi and Hans Ulrich Simon, editors, *Learning Theory, 19th Annual Conference on Learning Theory, COLT 2006, Pittsburgh, PA, USA, June 22-25, 2006, Proceedings*, volume 4005 of *Lecture Notes in Computer Science*, pages 605–619. Springer, 2006.

Nick Craswell. Mean reciprocal rank. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, page 1703. Springer US, 2009a.

Nick Craswell. Mean average precision. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, page 1703. Springer US, 2009b.

Fabio Crestani, Mounia Lalmas, C. J. van Rijsbergen, and Iain Campbell. "is this document relevant? ... probably": A survey of probabilistic models in information retrieval. *ACM*

*Comput. Surv.*, 30(4):528–552, 1998.

W. Bruce Croft. Document representation in probabilistic models of information retrieval. *J. Am. Soc. Inf. Sci.*, 32(6):451–457, 1981.

W. Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines - Information Retrieval in Practice*. Pearson Education, 2009.

Zhuyun Dai and Jamie Callan. Deeper text understanding for IR with contextual neural language modeling. In Benjamin Piwowarski, Max Chevalier, Éric Gaussier, Yoelle Maarek, Jian-Yun Nie, and Falk Scholer, editors, *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*, pages 985–988. ACM, 2019.

Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, pages 126–134, 2018.

Luis M. de Campos, Juan M. Fernández-Luna, and Juan F. Huete. Reducing propagation effort in large polytrees: An application to information retrieval. In José A. Gámez and Antonio Salmerón, editors, *First European Workshop on Probabilistic Graphical Models, 6-8 November - 2002 - Cuenca (Spain), Electronic Proceedings*, 2002.

Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407, 1990.

Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W. Bruce Croft. Neural ranking models with weak supervision. In Noriko Kando, Tetsuya Sakai, Hideo Joho, Hang Li, Arjen P. de Vries, and Ryen W. White, editors, *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 65–74. ACM, 2017.

Ron Van den Branden. Introduction to modern information retrieval. second editiong. g. chowdhury. *Lit. Linguistic Comput.*, 22(1):112–115, 2007.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

M. Carl Drott. Information retrieval systems: Theory and implementation, by gerald kowalski. *J. Am. Soc. Inf. Sci.*, 49(7):668–670, 1998.

Ofer Egozi, Shaul Markovitch, and Evgeniy Gabrilovich. Concept-based information retrieval using explicit semantic analysis. *ACM Trans. Inf. Syst.*, 29(2):8:1–8:34, 2011.

Joel L Fagan. Experiments in automatic phrase indexing for document retrieval: a comparison of syntactic and non-syntactic methods. Technical report, Cornell University, 1987.

Yixing Fan, Jiafeng Guo, Yanyan Lan, Jun Xu, Chengxiang Zhai, and Xueqi Cheng. Modeling diverse relevance patterns in ad-hoc retrieval. In Kevyn Collins-Thompson, Qiaozhu Mei, Brian D. Davison, Yiqun Liu, and Emine Yilmaz, editors, *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*, pages 375–384. ACM, 2018.

Hui Fang, Tao Tao, and ChengXiang Zhai. A formal study of information retrieval heuristics. In Mark Sanderson, Kalervo Järvelin, James Allan, and Peter Bruza, editors, *SIGIR 2004: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Sheffield, UK, July 25-29, 2004*, pages 49–56. ACM, 2004.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

Norbert Fuhr. Probabilistic models in information retrieval. *Comput. J.*, 35(3):243–255, 1992.

Yoav Goldberg and Omer Levy. word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649. IEEE, 2013.

Jian Gu, Guang-Hua Yin, Pengfei Huang, Jinlu Guo, and Lijun Chen. An improved back propagation neural network prediction model for subsurface drip irrigation system. *Computers & Electrical Engineering*, 60:58–65, 2017.

Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In Snehasis Mukhopadhyay, ChengXiang Zhai, Elisa Bertino, Fabio Crestani, Javed Mostafa, Jie Tang, Luo Si, Xiaofang Zhou, Yi Chang, Yunyao Li, and Parikshit Sondhi, editors, *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 55–64. ACM, 2016.

Christophe Van Gysel, Evangelos Kanoulas, and Maarten de Rijke. Lexical query modeling in session search. In Ben Carterette, Hui Fang, Mounia Lalmas, and Jian-Yun Nie, editors, *Proceedings of the 2016 ACM on International Conference on the Theory of Information Retrieval, ICTIR 2016, Newark, DE, USA, September 12- 6, 2016*, pages 69–72. ACM, 2016.

Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA,*

*June 7-12, 2015*, pages 5353–5360. IEEE Computer Society, 2015.

Robert Hecht-Nielsen. Theory of the backpropagation neural network. *Neural Networks*, 1 (Supplement-1):445–448, 1988.

Monika Rauch Henzinger. Link analysis in web information retrieval. *IEEE Data Eng. Bull.*, 23(3):3–8, 2000.

Djoerd Hiemstra. *Using language models for information retrieval*. Univ. Twente, 2001. ISBN 978-90-75296-05-1.

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

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. 2001.

Sebastian Hofstätter, Markus Zlabinger, and Allan Hanbury. Interpretable & time-budget-constrained contextualization for re-ranking. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 513–520. IOS Press, 2020.

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2042–2050, 2014.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. Learning deep structured semantic models for web search using clickthrough data. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 2333–2338, 2013.

Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

Karen Sparck Jones, Steve Walker, and Stephen E. Robertson. A probabilistic model of information retrieval: development and comparative experiments - part 1. *Inf. Process. Manag.*, 36(6):779–808, 2000a.

Karen Sparck Jones, Steve Walker, and Stephen E. Robertson. A probabilistic model of information retrieval: development and comparative experiments - part 2. *Inf. Process. Manag.*, 36(6):809–840, 2000b.

Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over BERT. In Jimmy Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu, editors, *Proceedings of the 43rd*

*International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 39–48. ACM, 2020.

Jungi Kim, Jinseok Nam, and Iryna Gurevych. Learning semantics with deep belief network for cross-language information retrieval. In *COLING 2012, 24th International Conference on Computational Linguistics, Proceedings of the Conference: Posters, 8-15 December 2012, Mumbai, India*, pages 579–588, 2012.

Youngjoong Ko. A study of term weighting schemes using class information for text classification. In William R. Hersh, Jamie Callan, Yoelle Maarek, and Mark Sanderson, editors, *The 35th International ACM SIGIR conference on research and development in Information Retrieval, SIGIR '12, Portland, OR, USA, August 12-16, 2012*, pages 1029–1030. ACM, 2012.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012.

Robert Krovetz. Viewing morphology as an inference process. *Artif. Intell.*, 118(1-2):277–294, 2000.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

Dik Lun Lee, Huei Chuang, and Kent E. Seamons. Document ranking and the vector-space model. *IEEE Softw.*, 14(2):67–75, 1997. doi: 10.1109/52.582976.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880. Association for Computational Linguistics, 2020.

Hang Li. *Learning to Rank for Information Retrieval and Natural Language Processing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2011.

Hong Liang, Xiao Sun, Yunlei Sun, and Yuan Gao. Text feature extraction based on deep learning: a review. *EURASIP J. Wirel. Commun. Netw.*, 2017:211, 2017.

Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. Distilling dense representations for ranking using tightly-coupled teachers. *CoRR*, abs/2010.11386, 2020. URL `https://arxiv.org/abs/2010.11386`.

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

Tie-Yan Liu. *Learning to Rank for Information Retrieval*. Springer, 2011. ISBN 978-3-642-14266-6. doi: 10.1007/978-3-642-14267-3. URL `https://doi.org/10.1007/978-3-642-14267-3`.

Wenhao Lu, Jian Jiao, and Ruofei Zhang. Twinbert: Distilling knowledge to twin-structured compressed BERT models for large-scale retrieval. In Mathieu d'Aquin, Stefan Dietze, Claudia Hauff, Edward Curry, and Philippe Cudré-Mauroux, editors, *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 2645–2652. ACM, 2020a.

Wenhao Lu, Jian Jiao, and Ruofei Zhang. Twinbert: Distilling knowledge to twin-structured BERT models for efficient retrieval. *CoRR*, abs/2002.06275, 2020b.

Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. Sparse, dense, and attentional representations for text retrieval. *Trans. Assoc. Comput. Linguistics*, 9:329–345, 2021.

Doaa Mabrouk, Sherine Rady, Nagwa Badr, and ME Khalifa. A survey on information retrieval systems' modeling using term dependencies and term weighting. In *2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS)*, pages 321–328. IEEE, 2017.

Craig Macdonald, Rodrygo L. T. Santos, and Iadh Ounis. On the usefulness of query features for learning to rank. In Xue-wen Chen, Guy Lebanon, Haixun Wang, and Mohammed J. Zaki, editors, *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 2559–2562. ACM, 2012.

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. Probabilistic information retrieval. *Introduction to Information Retrieval*, pages 220–235, 2009.

Irina Matveeva, Chris Burges, Timo Burkard, Andy Laucius, and Leon Wong. High accuracy retrieval with multiple nested ranker. In Efthimis N. Efthimiadis, Susan T. Dumais, David Hawking, and Kalervo Järvelin, editors, *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006*, pages 437–444. ACM, 2006.

Oren Melamud, Jacob Goldberger, and Ido Dagan. context2vec: Learning generic context embedding with bidirectional LSTM. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*, pages 51–61, 2016.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

Bhaskar Mitra and Nick Craswell. An updated duet model for passage re-ranking. *CoRR*, abs/1903.07666, 2019.

Bhaskar Mitra, Fernando Diaz, and Nick Craswell. Learning to match using local and distributed representations of text for web search. In Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich, editors, *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 1291–1299. ACM, 2017.

Mandar Mitra, Chris Buckley, Amit Singhal, and Claire Cardie. An analysis of statistical and syntactic phrases. In *Computer-Assisted Information Retrieval (Recherche d'Information et ses Applications) - RIAO 1997, 5th International Conference, McGill University, Montreal, Canada, June 25-27, 1997. Proceedings*, pages 200–217, 1997.

Ramesh Nallapati and James Allan. Capturing term dependencies using a language model based on sentence trees. In *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA, November 4-9, 2002*, pages 383–390. ACM, 2002.

Yifan Nie, Alessandro Sordoni, and Jian-Yun Nie. Multi-level abstraction convolutional model with weak supervision for information retrieval. In Kevyn Collins-Thompson, Qiaozhu Mei, Brian D. Davison, Yiqun Liu, and Emine Yilmaz, editors, *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*, pages 985–988. ACM, 2018.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, and Xueqi Cheng. A study of matchpyramid models on ad-hoc retrieval. *CoRR*, abs/1606.04648, 2016a.

Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. Text matching as image recognition. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2793–2799. AAAI Press, 2016b.

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 Ee-Peng Lim, Marianne Winslett, Mark Sanderson, Ada Wai-Chee Fu, Jimeng Sun, J. Shane Culpepper, Eric Lo, Joyce C. Ho, Debora Donato, Rakesh Agrawal, Yu Zheng, Carlos Castillo, Aixin Sun, Vincent S. Tseng, and Chenliang Li, editors, *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, pages 257–266. ACM, 2017.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and*

*Conference Proceedings*, pages 1310–1318. JMLR.org, 2013.

Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In Xiaohua Jia, editor, *Proceedings of the 1st International Conference on Scalable Information Systems, Infoscale 2006, Hong Kong, May 30-June 1, 2006*, volume 152 of *ACM International Conference Proceeding Series*, page 1. ACM, 2006.

Rama Kumar Pasumarthi, Xuanhui Wang, Cheng Li, Sebastian Bruch, Michael Bendersky, Marc Najork, Jan Pfeifer, Nadav Golbandi, Rohan Anil, and Stephan Wolf. Tf-ranking: Scalable tensorflow library for learning-to-rank. *CoRR*, abs/1812.00073, 2018.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

Enoch Peserico and Luca Pretto. Score and rank convergence of HITS. In James Allan, Javed A. Aslam, Mark Sanderson, ChengXiang Zhai, and Justin Zobel, editors, *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, Boston, MA, USA, July 19-23, 2009*, pages 770–771. ACM, 2009.

MARTIN F Porter. Implementing a probabilistic information retrieval system. *Information Technology: Research and Development*, 1(2):131–156, 1982.

Yifan Qiao, Chenyan Xiong, Zhenghao Liu, and Zhiyuan Liu. Understanding the behaviors of BERT in ranking. *CoRR*, abs/1904.07531, 2019. URL `http://arxiv.org/abs/1904.07531`.

Tao Qin and Tie-Yan Liu. Introducing LETOR 4.0 datasets. *CoRR*, abs/1306.2597, 2013.

Chen Qu, Liu Yang, Minghui Qiu, W. Bruce Croft, Yongfeng Zhang, and Mohit Iyyer. BERT with history answer embedding for conversational question answering. In Benjamin Piwowarski, Max Chevalier, Éric Gaussier, Yoelle Maarek, Jian-Yun Nie, and Falk Scholer, editors, *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*, pages 1133–1136. ACM, 2019.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI*, 1(8):9, 2019.

Vijay V. Raghavan and S. K. Michael Wong. A critical analysis of vector space model for information retrieval. *J. Am. Soc. Inf. Sci.*, 37(5):279–287, 1986.

Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, 2003.

Stephen E. Robertson. Evaluation in information retrieval. In Maristella Agosti, Fabio Crestani, and Gabriella Pasi, editors, *Lectures on Information Retrieval, Third European Summer-School, ESSIR 2000, Varenna, Italy, September 11-15, 2000, Revised Lectures*, volume 1980 of *Lecture Notes in Computer Science*, pages 81–92. Springer, 2000.

Stephen E. Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

Stephen E. Robertson, Hugo Zaragoza, and Michael J. Taylor. Simple BM25 extension to multiple weighted fields. In David A. Grossman, Luis Gravano, ChengXiang Zhai, Otthein Herzog, and David A. Evans, editors, *Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004*, pages 42–49. ACM, 2004.

Thomas Roelleke. *Information Retrieval Models: Foundations & Relationships*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers, 2013.

Gerard Salton. *The SMART retrieval system—experiments in automatic document processing*. Prentice-Hall, Inc., 1971.

Abhinav Sethy and Bhuvana Ramabhadran. Bag-of-word normalized n-gram models. In *INTERSPEECH 2008, 9th Annual Conference of the International Speech Communication Association, Brisbane, Australia, September 22-26, 2008*, pages 1594–1597. ISCA, 2008.

Aliaksei Severyn and Alessandro Moschitti. Learning to rank short text pairs with convolutional deep neural networks. In Ricardo Baeza-Yates, Mounia Lalmas, Alistair Moffat, and Berthier A. Ribeiro-Neto, editors, *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, August 9-13, 2015*, pages 373–382. ACM, 2015.

Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. Learning semantic representations using convolutional neural networks for web search. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*, pages 373–374. ACM, 2014.

Grigori Sidorov, Alexander F. Gelbukh, Helena Gómez-Adorno, and David Pinto. Soft similarity and soft cosine measure: Similarity of features in vector space model. *Computación y Sistemas*, 18(3), 2014.

Ashutosh Kumar Singh and P Ravi Kumar. A comparative study of page ranking algorithms for information retrieval. *Int. J. Electr. Comput. Eng*, 4:469–480, 2009.

Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24 (4):35–43, 2001.

Amit Singhal, Chris Buckley, and Mandar Mitra. Pivoted document length normalization. In Hans-Peter Frei, Donna Harman, Peter Schäuble, and Ross Wilkinson, editors, *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'96, August 18-22, 1996, Zurich, Switzerland (Special Issue of the SIGIR Forum)*, pages 21–29. ACM, 1996.

Hajung Sohn and Hyunju Lee. MC-BERT4HATE: hate speech detection using multi-channel BERT for different languages and translations. In Panagiotis Papapetrou, Xueqi Cheng, and Qing He, editors, *2019 International Conference on Data Mining Workshops, ICDM Workshops 2019, Beijing, China, November 8-11, 2019*, pages 551–559. IEEE, 2019.

Fei Song and W. Bruce Croft. A general language model for information retrieval. In *Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management, Kansas City, Missouri, USA, November 2-6, 1999*, pages 316–321. ACM, 1999.

Pascal Soucy and Guy W. Mineau. Beyond TFIDF weighting for text categorization in the vector space model. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 1130–1135. Professional Book Center, 2005.

Yue-Heng Sun, Pi-Lian He, and Zhi-Gang Chen. An improved term weighting scheme for vector space model. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE)*, volume 3, pages 1692–1695. IEEE, 2004.

Ming Tan, Tian Xia, Lily Guo, and Shaojun Wang. Direct optimization of ranking measures for learning to rank models. In *SIGKDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 856–864, 2013.

Tao Tao, Xuanhui Wang, Qiaozhu Mei, and ChengXiang Zhai. Language model information retrieval with document expansion. In Robert C. Moore, Jeff A. Bilmes, Jennifer Chu-Carroll, and Mark Sanderson, editors, *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 4-9, 2006, New York, New York, USA*. The Association for Computational Linguistics, 2006.

Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66, 2001.

Betty van Aken, Benjamin Winter, Alexander Löser, and Felix A. Gers. How does BERT answer questions?: A layer-wise analysis of transformer representations. In Wenwu Zhu, Dacheng Tao, Xueqi Cheng, Peng Cui, Elke A. Rundensteiner, David Carmel, Qi He, and Jeffrey Xu Yu, editors, *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, pages 1823–1832. ACM, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long*

*Beach, CA, USA*, pages 5998–6008, 2017.

François B. Vialatte and Andrzej Cichocki. Split-test bonferroni correction for QEEG statistical maps. *Biol. Cybern.*, 98(4):295–303, 2008.

Hanna M. Wallach. Topic modeling: beyond bag-of-words. In William W. Cohen and Andrew W. Moore, editors, *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, volume 148 of *ACM International Conference Proceeding Series*, pages 977–984. ACM, 2006.

Shengxian Wan, Yanyan Lan, Jun Xu, Jiafeng Guo, Liang Pang, and Xueqi Cheng. Matchsrnn: Modeling the recursive matching structure with spatial RNN. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 2922–2928. IJCAI/AAAI Press, 2016.

Xuanhui Wang, Cheng Li, Nadav Golbandi, Michael Bendersky, and Marc Najork. The lambdaloss framework for ranking metric optimization. In Alfredo Cuzzocrea, James Allan, Norman W. Paton, Divesh Srivastava, Rakesh Agrawal, Andrei Z. Broder, Mohammed J. Zaki, K. Selçuk Candan, Alexandros Labrinidis, Assaf Schuster, and Haixun Wang, editors, *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*, pages 1313–1322. ACM, 2018.

Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. A theoretical analysis of NDCG type ranking measures. In Shai Shalev-Shwartz and Ingo Steinwart, editors, *COLT 2013 - The 26th Annual Conference on Learning Theory, June 12-14, 2013, Princeton University, NJ, USA*, volume 30 of *JMLR Workshop and Conference Proceedings*, pages 25–54. JMLR.org, 2013.

S. K. Michael Wong, Wojciech Ziarko, and P. C. N. Wong. Generalized vector space model in information retrieval. In *Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval, Montréal, Québec, Canada, June 5-7, 1985*, pages 18–25, 1985.

Ho Chung Wu, Robert Wing Pong Luk, Kam-Fai Wong, and Kui-Lam Kwok. Interpreting TF-IDF term weights as making relevance decisions. *ACM Trans. Inf. Syst.*, 26(3):13:1–13:37, 2008.

Qiang Wu, Christopher J. C. Burges, Krysta Marie Svore, and Jianfeng Gao. Adapting boosting for information retrieval measures. *Inf. Retr.*, 13(3):254–270, 2010.

Yu Wu, Wei Wu, Chen Xing, Ming Zhou, and Zhoujun Li. Sequential matching network: A new architecture for multi-turn response selection in retrieval-based chatbots. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 496–505. Association for Computational Linguistics, 2017.

Chenyan Xiong, Jamie Callan, and Tie-Yan Liu. Word-entity duet representations for document ranking. In Noriko Kando, Tetsuya Sakai, Hideo Joho, Hang Li, Arjen P. de Vries, and Ryen W. White, editors, *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 763–772. ACM, 2017a.

Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. End-to-end neural ad-hoc ranking with kernel pooling. In Noriko Kando, Tetsuya Sakai, Hideo Joho, Hang Li, Arjen P. de Vries, and Ryen W. White, editors, *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 55–64. ACM, 2017b.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

Liu Yang, Qingyao Ai, Jiafeng Guo, and W. Bruce Croft. anmm: Ranking short answer texts with attention-based neural matching model. In Snehasis Mukhopadhyay, ChengXiang Zhai, Elisa Bertino, Fabio Crestani, Javed Mostafa, Jie Tang, Luo Si, Xiaofang Zhou, Yi Chang, Yunyao Li, and Parikshit Sondhi, editors, *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 287–296. ACM, 2016.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 5754–5764, 2019.

Andrew Yates, Rodrigo Nogueira, and Jimmy Lin. Pretrained transformers for text ranking: BERT and beyond. In Liane Lewin-Eytan, David Carmel, Elad Yom-Tov, Eugene Agichtein, and Evgeniy Gabrilovich, editors, *WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining, Virtual Event, Israel, March 8-12, 2021*, pages 1154–1156. ACM, 2021.

Emine Yilmaz, Evangelos Kanoulas, and Javed A Aslam. A simple and efficient sampling method for estimating ap and ndcg. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 603–610. ACM, 2008.

ChengXiang Zhai. Statistical language models for information retrieval: A critical review. *Foundations and Trends in Information Retrieval*, 2(3):137–213, 2008.

ChengXiang Zhai and John D. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In W. Bruce Croft, David J. Harper, Donald H. Kraft, and Justin Zobel, editors, *SIGIR 2001: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, September 9-13, 2001, New Orleans, Louisiana, USA*, pages 334–342. ACM, 2001.

Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. Learning to retrieve: How to train a dense retrieval model effectively and efficiently. *CoRR*, abs/2010.10469, 2020.

Ye Zhang, Matthew Lease, and Byron C. Wallace. Active discriminative text representation learning. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 3386–3392. AAAI Press, 2017a.

Yizhe Zhang, Dinghan Shen, Guoyin Wang, Zhe Gan, Ricardo Henao, and Lawrence Carin. Deconvolutional paragraph representation learning. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 4169–4179, 2017b.

Xiaoqiang Zhou, Baotian Hu, Jiaxin Lin, Yang Xiang, and Xiaolong Wang. ICRC-HIT: A deep learning based comment sequence labeling system for answer selection challenge. In Daniel M. Cer, David Jurgens, Preslav Nakov, and Torsten Zesch, editors, *Proceedings of the 9th International Workshop on Semantic Evaluation, SemEval@NAACL-HLT 2015, Denver, Colorado, USA, June 4-5, 2015*, pages 210–214. The Association for Computer Linguistics, 2015.

Justin Zobel and Alistair Moffat. Exploring the similarity space. *SIGIR Forum*, 32(1):18–34, 1998.

Keneilwe Zuva and Tranos Zuva. Evaluation of information retrieval systems. *International journal of computer science & information technology*, 4(3):35, 2012.