A Position-Aware Deep Model for Relevance Matching in Information Retrieval

Kai Hui
Max Planck Institute for Informatics
khui@mpi-inf.mpg.de

Andrew Yates
Max Planck Institute for Informatics
ayates@mpi-inf.mpg.de

Klaus Berberich
Max Planck Institute for Informatics
kberberi@mpi-inf.mpg.de

Gerard de Melo
Rutgers University
gdm@demelo.org

ABSTRACT

In order to adopt deep learning for ad-hoc information retrieval, it is essential to establish suitable representations of query-document pairs and to design neural architectures that are able to digest such representations. In particular, they ought to capture all relevant information required to assess the relevance of a document for a given user query, including term overlap as well as positional information such as proximity and term dependencies. While previous work has successfully captured unigram term matches, none has successfully used position-dependent information on a standard benchmark test collection. In this work, we address this gap by encoding the relevance matching in terms of similarity matrices and using a deep model to digest such matrices. We present a novel model architecture consisting of convolutional layers to capture term dependencies and proximity among query term occurrences, followed by a recurrent layer to capture relevance over different query terms. Extensive experiments on Trec Web Track data confirm that the proposed model with similarity matrix representations yields improved search results.

CCS CONCEPTS

•**Information systems** →**Retrieval models and ranking;** *Web searching and information discovery;*

1 INTRODUCTION

Deep learning has shown enormous potential in the last few years and has brought significant advances in fields such as natural language processing and computer vision. Whether similar advances are also possible for ad-hoc information retrieval remains an open question that has only recently begun to be investigated. The key challenges here are 1) to establish suitable representations for queries and documents (or query-document pairs), and 2) to design a neural model architecture that is well-suited for taking advantage of such representations.

Looking back at decades of research on ad-hoc information retrieval, two ideas have received ample attention and proven useful for modeling the relationship between queries and documents. The first is to not only rely on query terms, but to also consider semantically similar terms – as best exemplified by query-expansion methods [7, 24]. Taking into account information about where query terms occur in a document and how they depend on each other is the second idea – as exploited by retrieval models aware of term proximity [22] or term dependencies [11, 14]. Little work has looked into combining both aforementioned considerations.

In fact, existing work on deep learning for ad-hoc information retrieval has often neglected these ideas. Early approaches, such as Shen et al. [10, 20], represented queries and documents separately in a common space, thereby losing valuable information such as which of the query terms were (closely) matched by the document. Inspired by traditional retrieval models, Guo et al. [8] more recently observed that a representation should retain information regarding the matching of particular query terms and of semantically similar terms. Their approach relies on similarity histograms that capture how often the query terms and semantically similar terms occur in a document. Experiments on Trec benchmarks showed that their proposed *DRMM* could outperform all prior deep retrieval models and, more importantly, was the only one able to outperform traditional retrieval models such as the query-likelihood model (*QL*) [25] on standard Trec benchmarks. Positional information, however, is discarded by their approach, making it impossible for the network architecture to pick up cues related to proximity or mutual dependency of terms. While the MatchPyramid [16] model does incorporate proximity information, it has been shown to perform substantially worse than DRMM on standard Trec benchmarks [8].

Motivated by the impact of positional information shown for traditional retrieval models, in this work we investigate how positional information can be retained in a well-performing deep learning model. We conjecture that deep retrieval models may be able to benefit from synergistic effects when jointly exploiting semantic similarity as well as positional cues or mutual dependencies. To this end, we present a novel approach called *PACRR*, short for *Position-Aware Convolutional-Recurrent Relevance Matching*. Our approach relies on similarity matrices that record the semantic similarity between each query term and each individual term occurring in a document.

Based on these similarity matrices, our neural architecture consists of a convolutional neural network (CNN) with multiple filters to extract local interactions between query terms and terms from the document. We rely on filters of different sizes to capture interactions corresponding to, for instance, bi-gram and tri-gram matches.

Conference'17, Washington, DC, USA

DOI:

Going beyond mere n-gram matching, as used in the Sequential Dependence Model [14], these filters actually also capture more elaborate matching patterns such as non-contiguous matches of query terms in a document. The information extracted by these filters can be regarded as a set of dozens or even hundreds of text snippets identified within a document, each of which appears relevant to the query. Following the convolutional layer, a k-max pooling [13] layer retains the k strongest signals along the query dimension, to focus only on the most promising text snippets.

Given these promising text snippets, the model needs to make an overall assessment of whether the document is relevant enough to satisfy the information need behind the query. A document is deemed more relevant if it contains text snippets covering the query in its entirety. As a concrete example, for the query "deep learning and information retrieval", a document with content related to both deep learning and to information retrieval should be preferred over one which extensively discusses only one of the two topics. Our model thus needs to account for the degree of relevance of the individual text snippets as well as the level of correspondence over all successive parts of the query. This is achieved via a recurrent layer that learns from the k strongest signals per individual query term and aggregates them to yield a single relevance score for each query-document pair.

In extensive experiments across six years (2009–14) of data from the Trec Web Track, we compare our approach against DRMM, the state-of-the-art deep learning model on Trec benchmarks. We observe that our approach significantly outperforms DRMM when re-ranking results by the query-likelihood model, as a relatively weak base ranker. On average, we can improve search results from the query-likelihood model by more than 50%, thus yielding a system that would have ranked among the top-3 systems in the Trec Web Track ad-hoc task 2012–14. Beyond that, the proposed model also does a better job when re-ranking results by participants' systems from Trec, consistently improving retrieval effectiveness by at least 20% average across different years in terms of nDCG@20.

Contributions. The contributions that we make in this work are threefold:

- We highlight and demonstrate the importance of preserving positional information in representations for deep relevance matching.
- We propose a novel deep model that incorporates insights from traditional information retrieval models, based on a custom-tailored position-preserving representation.
- In extensive experiments, we compare our approach on all available Trec Web Track judgments for the ad-hoc task on ClueWeb09 and ClueWeb12, obtaining consistently superior results for our proposed model.

Organization. The rest of this paper unfolds as follows. Section 2 gives a more complete overview of related work and puts our approach in context. Section 3 describes our approach for computing similarity matrices and the architecture of our deep learning model. The setup and results of our extensive experimental evaluation can be found in Section 4. Finally, we conclude in Section 5.

2 RELATED WORK

Ad-hoc retrieval systems aim at ranking documents with respect to their relevance relative to given queries. Recently, the promise of deep learning as a potential driver for new advances in retrieval quality has attracted significant attention. Early works like Deep Structured Semantic Models [10] (*DSSM*) learned low-dimensional representations of queries and documents in a shared space and performed ranking by comparing the cosine similarity between a given query's representation and the representations of documents in the collection. Similar approaches such as C-DSSM [21] relied on alternative means of learning document representations. Severyn and Moschitti [19] further combined learned semantic representations with external features to rank question answers and tweets. In addition, other approaches to learning representation of documents or queries can also be deployed for retrieval, by taking the similarity between such representations as relevance scores. Models of this sort include ARC-I and ARC-II by Pang et al. [17], which performed text classification by producing representations of documents and queries separately (ARC-I), or, alternatively, produced representations that also considered the interactions between queries and documents jointly (ARC-II). The experiments from [8] (as well as our own pilot experiments) showed that none of the above deep models can consistently improve traditional retrieval models such as *QL* on the long-established Trec benchmarks.

Guo et al. [8] highlighted that the matching needed for information retrieval differs from the kind that is used in NLP tasks, which typically aim at *semantic matching*. Information retrieval ranking models, in contrast, are concerned with *relevance matching*. The former focuses on comparing the meaning of two input texts, while the latter focuses on the inherently asymetric goal of representing the text and determining its relevance to a user query. The overall semantics of the document needn't be similar to the user intent expressed by the query. Indeed, traditional retrieval models such as query-likelihood (*QL*) [25] are heavily based on this notion of relevance matching, capturing it via unigram occurrences of query terms. *DRMM* learns the same sort of unigram matches as in traditional retrieval models, but with more advanced instruments from deep learning. In particular, *DRMM* takes a sequence of fixed-length query term similarity histograms as input; each histogram h_j represents the matches between one query term q_j in a given query q and the terms in a given document. The query similarity histograms are each fed through a series of fully connected layers to produce a similarity signal for each query term. The document's relevance score $rel(q, d)$ is a weighted summation of each query term's similarity signal. *DRMM* was evaluated on Trec Robust Track 2004 and the Web Track 2009–11. The experiments demonstrated that *DRMM* outperformed both *DSSM* and several deep models developed for NLP tasks. In fact, *DRMM* outperformed the strongest *DSSM* variant by a minimum of 68% nDCG@20, and we observed similar results in our pilot experiments. Thus, we consider *DRMM* as the main baseline to compare against, as it is the state-of-the-art deep retrieval model based on results over the Trec Web Track ad-hoc benchmark. Another recent work called Duet [15] proposes a deep ranking model that considers both exact matches between document and query terms (the *local model*) and the similarity between low-dimensional representations of the query and

document (the *distributed model*). The authors evaluated Duet on queries from Bing search logs with 199,753 training examples, and reported nDCG@10 increases over *DRMM* of 18% (weighted by query occurrences) and 2% (unweighted). Interestingly, *DRMM* did not substantially outperform *QL* in this comparison, whereas Guo et al. saw large increases in performance in their experiments on Trec Web Track data. We remark that the weighted evaluation set appears incomparable with the Trec Web Track. In particular, the test queries from the Trec Web Track include a significant proportion of more focused topics designed to represent more specific, less frequent, and possibly more difficult queries [6], whereas the weighted test data places more emphasis on frequent head queries. Given that the 1 million documents and 200,000 Bing queries used to train Duet are far beyond what is available to us, we compare our methods with Duet indirectly, by examining the relative improvements over *QL* and *DRMM*.

Different from *DRMM* [8] and Duet [15], our study seeks to show that, beyond unigram matching, positional information such as term dependencies [11, 14] and query term proximity, which have also been widely used in traditional retrieval models, can lead to significant improvements. In addition, as mentioned by Cohen et al. [5], there are two sources of difficulty in applying neural retrieval models to ad-hoc retrieval: variations in document lengths and variations in relevance granularity. Thus, we aim at addressing varying document lengths with a novel deep retrieval model, which additionally learns from positional information between a query and document pair.

3 METHOD

We now describe our novel *PACRR* approach, which consists of two main parts: a relevance matching component that converts each query-document pair into a similarity matrix $sim_{|q|\times|d|}$ and a deep architecture that takes a given query-document similarity matrix as input and produces a query-document relevance score $rel(q, d)$. The pipeline is summarized in Figure. 1. First, the relevance matching component transforms each query q and document d into a query-document similarity matrix $sim_{|q|\times|d|}$ by comparing each query term to each document term and recording the similarity scores. Given such a similarity matrix, our deep model then applies convolutional (CNN) layers to compare query n-grams with document n-grams, followed by max pooling operations to identify the strongest signals for each query term and n-gram size. Finally, a recurrent (RNN) layer aggregates the available relevance signals across different query terms into an overall query-document relevance score $rel(q, d)$.

3.1 Relevance Matching

We propose encoding the query-document relevance matching via query-document similarity matrices $sim_{|q|\times|d|}$ that encodes the similarity between query and document terms. In particular, given a document d and a query q, the similarity between every term pair from d and q is encoded as a similarity matrix $sim_{|q|\times|d|}$, where sim_{ij} corresponds to the similarity between the i-th term from the query q and the j-th term from the document d. When using cosine similarity, we have $sim \in [-1, 1]^{|q|\times|d|}$. Our similarity matrix approach retains a richer signal than the similarity histogram approach used in prior work [8], which is limited to performing relevance matching against unigrams. Our matrices preserve both n-gram relevance signals and query coverage information. In particular, n-gram matching corresponds to consecutive documents terms that are highly similar to at least one of the query terms, while query coverage is reflected in the number of rows in *sim* that include at least one cell with high similarity. As in [8], the similarity between a query term q and document term d is calculated by taking the cosine similarity between the terms' *word2vec* vectors.

The subsequent processing in PACRR's convolutional layers requires that each query-document similarity matrix have the same dimensionality. Given that the lengths of queries and documents vary, we first transform the raw similarity matrices $sim_{|q|\times|d|}$ into $sim_{l_q\times l_d}$ matrices with uniform l_q and l_d as the number of rows and columns. We unify the query dimension l_q by zero padding it to the maximum query length. With regard to the document dimension l_d, we describe two strategies, *firstk* and *kwindow*.

PACRR-firstk. The simplest approach is to zero pad the document dimension of each similarity matrix to the maximum document length. Akin to [15], the *firstk* distillation method simply keeps the first k columns in the matrix, which correspond to the first k terms in the document. If $k > |d|$ the remaining columns are zero padded. The *firstk* method is equivalent to zero padding if k is set to the maximum document length.

PACRR-kwindow. As mentioned in [8], the relevance matching is local. Document terms that have a low query similarity relative to a document's other terms cannot contribute to the document relevance score, so ignoring them has no effect on the final relevance score. Put differently, the relevance matching can be extracted in terms of phrases, sentences or pieces of text that include relevance information. Therefore, one can segment the documents according to the relevance relative to the given query, only retaining the text that is highly relevant to the given query. Given this observation, we prune query-document similarity cells with a low similarity score. In the case of unigrams, we simply choose the top l_d terms with the highest similarity to query terms.

In the case of n-grams, we produce a similarity matrix $sim^n_{l_q\times l_d}$ for each query-document pair and each n-gram size, because the n-gram size must be considered when the top n-term windows are chosen. For each n-term window in the document, *kwindow* calculates the maximum similarity between each term and the query terms, and then calculates the average similarity over each n-term window. It then selects the top $k = \lfloor l_d/n \rfloor$ windows by average similarity and discards all other terms in the document. The document dimension is zero padded if $\lfloor l_d/n \rfloor$ is not a multiple of k. When the convolutional layer later operates on a similarity matrix produced by *kwindow*, the model's stride is set to n (i.e., the sliding window moves ahead n terms at a time rather than one term at a time) so that it only considers consecutive n-grams that were present in the original document. This variant's output is a similarity matrix $sim^n_{l_q\times l_d}$ for each n-gram size. These matrices support n-gram relevance matching by retaining the context around each high-similarity term.

Distillation example. Suppose that we target $l_d = 4$ and $l_q = 3$, given a two term query and a document with six terms. The

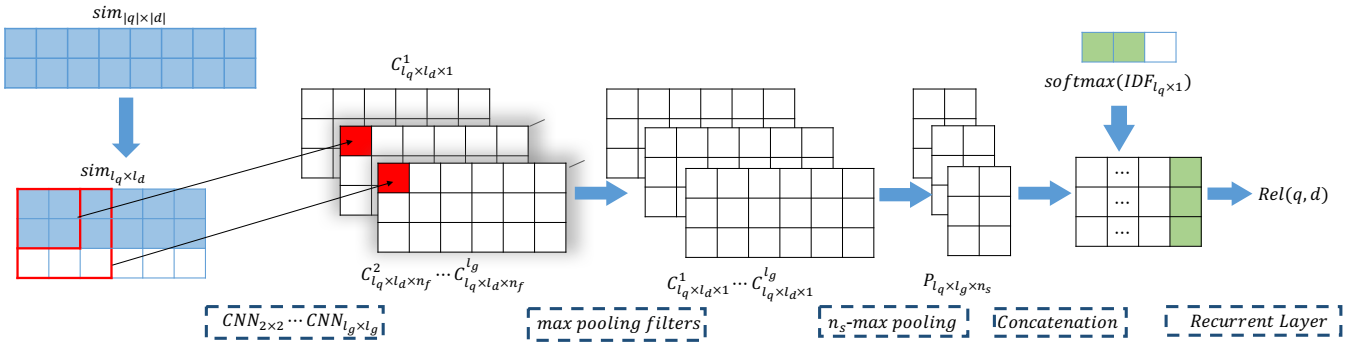

Figure 1: The pipeline of *PACRR*. The relevance matching component first transforms each query q and document d into a query-document similarity matrix $sim_{|q|\times|d|}$. Next, a distillation method transforms the raw similarity matrix into a similarity matrix with unified dimensions $sim_{l_q\times l_d}$. The *firstk* distillation method is displayed, and the same unified similarity matrix is used for all n-gram sizes. $l_g - 1$ convolutional layers (CNN) are applied to the distilled similarity matrices, $l_g = 3$ is displayed thus CNN with kernel size 2 and 3 are applied. Next, max pooling is applied, ending up with l_g matrices $C^1 \cdots C^{l_g}$. Following this, n_s-max pooling captures the strongest n_s signals over each query term and n-gram size, and the case when $n_s = 2$ is displayed. Finally, the similarity signals from different n-gram sizes are concatenated together, the query terms' normalized IDFs are added, and a recurrent layer combines these signals for each query term into a query-document relevance score $rel(q, d)$.

similarity is computed among query-document term pairs, resulting in a corresponding $sim_{2\times 6}$ similarity matrix such as

$$sim_{|q|\times|d|} = \begin{pmatrix} 0.9 & 0 & 0.7 & 0.1 & 0.2 & 0 \\ 0.1 & -0.1 & -0.5 & 0.8 & 0 & 0 \end{pmatrix}$$

Regardless of the chosen strategy, we zero pad the query dimension because $l_q > |q|$. The *firstk* variant then distills the similarity matrix by simply taking the first l_d columns regardless of the n-gram size:

$$sim_{l_q\times l_d} = \begin{pmatrix} 0.9 & 0 & 0.7 & 0.1 \\ 0.1 & -0.1 & -0.5 & 0.8 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

We first illustrate the output of *kwindow* in the unigram case (i.e., $n = 1$). Recall that each column corresponds to a document term. Thus the max similarity for each document term is $[0.9, 0, 0.7, 0.8, 0.2, 0]$. We have $l_d = 4$, so the columns corresponding to the top four terms are kept and the rest discarded. This results in the document columns 1, 3, 4, and 5 being retained:

$$sim^1_{l_q\times l_d} = \begin{pmatrix} 0.9 & 0.7 & 0.1 & 0.2 \\ 0.1 & -0.5 & 0.8 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

In the n-gram case, we must preserve n-term windows. Suppose we set $n = 2$. Taking the max query term similarity for each document term and using it to calculate the average similarity over each two term window, we obtain $[0.45, 0.35, 0.75, 0.5, 0.1]$. $\lfloor l_d/n \rfloor = 2$, so the top two windows are selected for inclusion in the similarity matrix. These correspond to the windows beginning at positions 3 and 4:

$$sim^2_{l_q\times l_d} = \begin{pmatrix} 0.7 & 0.1 & 0.1 & 0.2 \\ -0.5 & 0.8 & 0.8 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Note that the term in position 4 of the original similarity matrix appears twice; this preserves the n-grams encountered in the original similarity matrix. The window over positions 3 and 4 corresponds to the first occurrence of term 4, and the window over positions 4 and 5 accounts for the second occurrence of term 4. While the term in position 1 of the original similarity matrix has the highest similarity of any query term (0.9), it is not useful for n-gram matching because the window that includes this term has a comparably low similarity. It will still be considered by the model, however, because it is present in the unigram matrix.

3.2 Deep Retrieval Model

Given a query-document similarity matrix $sim_{l_q\times l_d}$ as input, our deep architecture then relies on convolutional layers to match query n-grams and document n-grams to produce a similarity signal for each query term, document term, and n-gram size n. Subsequently, max pooling layers extract the document's strongest similarity cue for each query term and each n. Finally, a recurrent layer aggregates these individual query term signals to predict a global query-document relevance score $rel(q, d)$.

Convolutional n-gram relevance matching. The purpose of this step is to match query n-grams with document n-grams given a query-document similarity matrix as input. This is accomplished by applying multiple two-dimensional convolutional layers with different kernel sizes to the input similarity matrix. Each convolutional layer is responsible for a specific n-gram size; by applying its kernel on $n \times n$ windows, it produces a similarity signal for each window. When the *firstk* method is used, each convolutional layer receives the same similarity matrix $sim_{l_q\times l_d}$ as input because *firstk* produces the same similarity matrix regardless of the n-gram size. When the *kwindow* method is used, each convolutional layer

receives a similarity matrix $sim^n_{l_q \times l_d}$ corresponding to the convolutional layer's n-gram size. We use $l_g - 1$ different convolutional layers with kernel sizes $2 \times 2, 3 \times 3, \ldots, l_g \times l_g$, corresponding to bi-gram, tri-gram, ..., l_g-gram matching, respectively, where the maximum n-gram size to consider is governed by a hyper-parameter l_g. The original similarity matrix corresponds to unigram matching, while a convolutional layer with kernel size $n \times n$ is responsible for capturing n-gram matching for an $n \times n$ square within $sim^n_{l_q \times l_d}$. Each convolutional layer applies n_f different filters to its input, where n_f is another hyper-parameter. We use a stride of size (1, 1) for the *firstk* distillation method, meaning that the convolutional kernel advances one step at a time in both the query and document dimensions. For the *kwindow* distillation method, we use a stride of (1, n) to move the convolutional kernel one step at a time in the query dimension, but n steps at a time in the document dimension. This ensures that the convolutional kernel only operates over consecutive terms that existed in the original document. Thus, we end up with $l_g - 1$ matrices $C^n_{l_q \times l_d \times n_f}$, where n indicates the corresponding n-gram size. The original similarity matrix will be used to handle unigrams.

Max pooling. The purpose of this step is to capture the n_s strongest similarity signals for each query term. Measuring the similarity signal separately for each query term allows the model to consider query term coverage, while capturing the n_s strongest similarity signals for each query term allows the model to consider signals from different kinds of relevance matching pattern, e.g., n-gram matching and non-contiguous matching. In practice, we use a small n_s to prevent the model from being biased by document length; while each similarity matrix contains the same number of document term scores, longer documents have more opportunities to contain terms that are similar to query terms.

To capture the strongest n_s similarity signals for each query term, we first perform max pooling over the filter dimension n_f to keep only the strongest signal from the n_f different filters. We then perform k-max pooling [13] over the query dimension l_q (corresponding to rows in the original similarity matrix) to keep the strongest n_s similarity signals for each query term. Both pooling steps are performed on each of the $l_g - 1$ matrices C^i from the convolutional layer (corresponding to each n-gram size) and on the original similarity matrix (corresponding to unigrams) to produce the 3-dimensional tensor $\mathcal{P}_{l_q \times l_g \times n_s}$. This tensor contains the n_s strongest similarity signals for each query term and for each n-gram size across all n_f filters.

Recurrent layer for global relevance. Finally, our model transforms the query term similarity signals in $\mathcal{P}_{l_q \times l_g \times n_s}$ into a single document relevance score $rel(q, d)$. It achieves this by applying a recurrent layer to $\mathcal{P}$, taking a sequence of vectors (i.e., one vector per query term) as input and learning weights to transform them into the final relevance score. In more detail, we first calculate the IDF of each query term q_i on the document collection and pass the IDFs through a softmax layer to normalize them relative to each other. Next, we split up the query term dimension to produce a matrix $\mathcal{P}_{l_g \times n_s}$ for each query term q_i. Finally, we form the recurrent layer's input by flattening each matrix $\mathcal{P}_{l_g \times n_s}$ into a vector by concatenating the matrix's rows together and appending query term q_i's normalized IDF onto the end of the vector. This sequence of vectors for each query term q_i is passed into a Long Short-Term Memory (LSTM) recurrent layer [9] with an output dimensionality of one. That is, the LSTM's input is a sequence of query term vectors where each vector is composed of the query term's normalized IDF and the similarity signals for the unigram case and for each n-gram size. The LSTM's output is then used as our document relevance score $rel(q, d)$.

Training objective and prediction. Our model is trained on triples consisting of a query q, relevant document d^+, and non-relevant document d^- using stochastic gradient descent (SGD) to minimize a loss function. We use a standard pairwise max margin loss as in Eq. 1.

$$\mathcal{L}(q, d^+, d^-; \Theta) = max(0, 1 - rel(q, d^+) + rel(q, d^-)) \quad (1)$$

At each training step, we perform SGD on a mini-batch of 32 triples. For the purpose of choosing the triples, we consider all documents that are judged with a label more relevant than *Rel*[1] as *highly relevant*, and put the remaining relevant documents into a *relevant* group. To pick each triple, we sample a relevance group with probability proportional to the number of documents in the group within the training set, and then we randomly sample a document with the chosen label to serve as the positive document d^+. If the chosen group is the highly relevant group, we randomly sample a document from the relevant group to serve as the negative document d^-. If the chosen group is the relevant group, we randomly sample a non-relevant document as d^-. This sampling procedure ensures that we differentiate between highly relevant documents (i.e., those with a relevance label of *HRel*, *Key* or *Nav*) and relevant documents (i.e., those are labeled as *Rel*). The training continues until a given number of epochs is reached. The model is saved at every epoch. We use the model with the best Err@20 on the validation set to make predictions.

4 EVALUATION

In this section, we empirically evaluate our novel PACRR methods and compare against both traditional query-likelihood (*QL*) and the deep retrieval model *DRMM* [8], which is the current state-of-the-art on the Trec Web Track ad-hoc task. The models are evaluated using manual relevance judgments by re-ranking search results from the Trec Web Track. First, we re-rank search results from query-likelihood relevance models [25]. The improvement relative to the one on *QL* is also used to compare with models from the literature, putting our model in context. After that, we scrutinize the models in more detail by re-ranking a wide range of runs from Trec Web Track participants from 2009–14. In practice, individual runs only cover a small subset of judged documents. Thus, we provide a further examination of predictions based on all available judgments by deriving pairwise comparisons among documents from Trec judgments. Finally, we discuss specific aspects such as accuracy over binary judgments.

[1]Judgments from Trec include junk pages (*Junk*), non-relevance (*NRel*), relevance (*Rel*), high relevance (*HRel*), key pages (*Key*) and navigational pages (*Nav*).

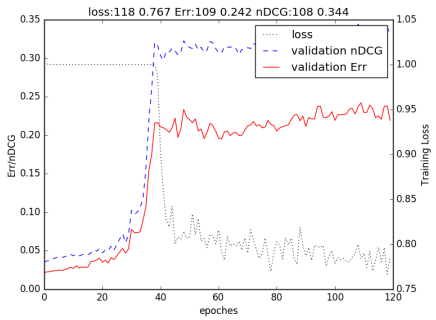

Figure 2: The loss on training data and the Err@20 and nDCG@20 per epoch on validation data when training on Web Track 2010–14. The x-axis is the epoch. The y-axis indicates the Err@20/nDCG@20 on the left and the training loss on the right. The best performance appears on 109th epoch with Err@20=0.242. Meanwhile, the lowest loss on training data occurs after 118 epoch, which equals 0.767.

4.1 Experimental Setup

We rely on the 2009–2014 TREC Web Track benchmarks[2], which are widely used in the IR community. The benchmarks are based on the CLUEWEB09[3] and CLUEWEB12[4] datasets as document collections, which include 500 and 700 million English web pages, respectively. In total, there are 300 queries and 113k judgments (qrels). Three years (2012–14) of query-likelihood baselines provided by TREC[5] serve as baseline runs in Section 4.2. In particular, two runs based on *QL* with pseudo-relevance feedback in each year are used: one based on all documents, referred to as *QL-ALL*, and one that relies on spam-filtering before ranking, referred to as *QL-NOSPAM* [6]. In Section 4.3, the runs submitted by participants from each year are also employed: there are 71 runs for 2009, 55 runs for 2010, 62 runs for 2011, 48 for 2012, 50 for 2013, and 27 for 2014. Err@20 [4] and nDCG@20 [12] serve as evaluation metrics, and both are computed with the script from TREC[6].

Training. For better comparisons and a more fine-grained analysis, we evaluate each year of TREC data separately. In particular, the available judgments are considered in accordance with the individual years of the Web Track, with 50 queries per year. Proceeding in a round-robin order, we report test results on one year by exploiting the respective remaining five years (250 queries) for training. From these 250 queries, we reserve 50 random queries as a held-out set for validation and hyper-parameter tuning, while the remaining 200 queries serve as the actual training set. As mentioned in Section 3, model parameters and trainings epochs are chosen by maximizing the Err@20 on the validation set. The selected model is then used to make predictions on the separate test data. One example for this training procedure is displayed in Figure 2. There are four hyper-parameters that govern the behavior of the proposed *PACRR-kwindow* and *PACRR-firstk*, the unified length of along the document dimension l_d, the k-max pooling size n_s, the maximum n-gram size l_g, and the number of filters used in convolutional layers n_f. Due to limited computational resources, we determine the range of hyper-parameters to try based on pilot experiments and domain intuitions. In particular, we evaluate $l_d \in [256, 384, 512, 640, 768]$, $n_s \in [1, 2, 3, 4]$ and $l_g \in [2, 3, 4]$. Given the limited possible matching patterns given a small kernel size, e.g., $l_g = 3$, the n_f is fixed to 32. For *PACRR-firstk*, we intuitively desire to retain as much information as possible from the input, and thus the l_d is always set to 768. The *DRMM* is trained in the same way, following the configuration of the model variants with best performance from [8], namely $DRMM_{LCH\times IDF}$.

[2]http://trec.nist.gov/tracks.html
[3]http://www.lemurproject.org/clueweb09.php/
[4]http://lemurproject.org/clueweb12/
[5]https://github.com/trec-web/trec-web-2014
[6]http://trec.nist.gov/data/web/12/gdeval.pl

4.2 Re-Ranking Query-Likelihood Results

We first examine the proposed model by re-ranking the search results from the *QL* baselines from TREC, which are only available on 2012–14. The results are summarized in Table 1. As reported in [8], *DRMM* significantly outperforms *QL-ALL* on WT12 and WT14. However, it only performs significantly better on WT12 when re-ranking *QL-NOSPAM*. Overall, we observe a 32% improvement on average with *DRMM* relative to either *QL* baseline. In comparison, Guo et. al [8] reported improvements of 5.2% and 28.4% when comparing against *QL* with different settings. Thus we argue that the performance of *DRMM* reported in this work is at least as good as the one from the original paper in this regard. As for the proposed models, we observe significant improvements with *PACRR-kwindow* on all three years when re-ranking *QL-ALL*, and significant improvements on 2012 and 2013 when re-ranking *QL-NOSPAM*, whereas *PACRR-firstk* improves both runs on all three years. More remarkably, by solely re-ranking the search results from *QL-ALL*, *PACRR-firstk* can already rank within the top-3 on all three years; and the re-ranked search results from *PACRR-kwindow* is also ranked within the top-5 based on nDCG@20. Moreover, from Table 1, when mixing the results from *QL-ALL* and *QL-NOSPAM* together, both *PACRR-kwindow* and *PACRR-firstk* provide around 60% improvements on average among years.

4.3 Re-Rank Search Results

Different models based on different methodologies lead to different initial document rankings, and *QL* is only one of them. In this section, we would like to examine the performance of the proposed models in re-ranking different sets of search results. Thus, we extend our analysis to re-ranked search results from all submitted runs in six years of the TREC Web Track ad-hoc task. In particular, the tested models make predictions for individual documents, which are used to re-rank the documents within each submitted run. Given that there are about 50 runs for each year, it is no longer feasible to list the metric scores for each re-ranked run. Thus, we summarize the results by comparing the performance of each run before and after re-ranking as in Figure 3. Additionally, we provide concrete statistics over each year to compare the methods under consideration. In Figure 3, each column corresponds to one year and the top row is based on Err@20, while the six plots at the bottom are based on nDCG@20. The axes represent the metric scores before (x-axis) and after (y-axis) re-ranking. The dashed line separating them is $y = x$, indicating when no change is observed from the

	year	#run	*PACRR-kwindow*				*PACRR-firstk*				DRMM				*QL*			
			Err@20	rank	nDCG@20	rank	Err@20	rank	nDCG@20	rank	Err@20	rank	nDCG@20	rank	Err@20	rank	nDCG@20	rank
QL-ALL	wt12	50	0.313 †	4	0.250 †‡	2	0.318 †	2	0.243 †‡	2	0.289 †	10	0.197 †	8	0.177	26	0.106	39
	wt13	52	0.139 †‡	14	0.279 †‡	4	0.166 †‡	3	0.295 †‡	3	0.124	25	0.228	20	0.101	38	0.190	36
	wt14	29	0.207 †	3	0.331 †	1	0.221 †	2	0.339 †	1	0.193 †	10	0.300 †	6	0.131	25	0.231	23
QL-NOSPAM	wt12	50	0.300 †‡	7	0.229 †‡	3	0.334 †‡	2	0.240 †‡	2	0.239	16	0.184 †	11	0.190	25	0.132	24
	wt13	52	0.124 †	25	0.241 †	14	0.143 †‡	13	0.261 †‡	7	0.105	35	0.212	25	0.095	46	0.180	40
	wt14	29	0.193	10	0.287 ‡	9	0.213 †	2	0.295 ‡	7	0.188	11	0.253	17	0.159	21	0.261	14

Table 1: Err@20 and nDCG@20 on Trec Web Track 2012–14 when re-ranking search results from *QL*. † indicates significantly different results relative to the *QL* baseline, and ‡ represents significantly different results compared with *DRMM*, using two-tailed paired Student's t-tests at a 95% confidence level. In addition, the relative ranks among all runs within the respective years according to Err@20 and nDCG@20 are also reported right after the absolute scores.

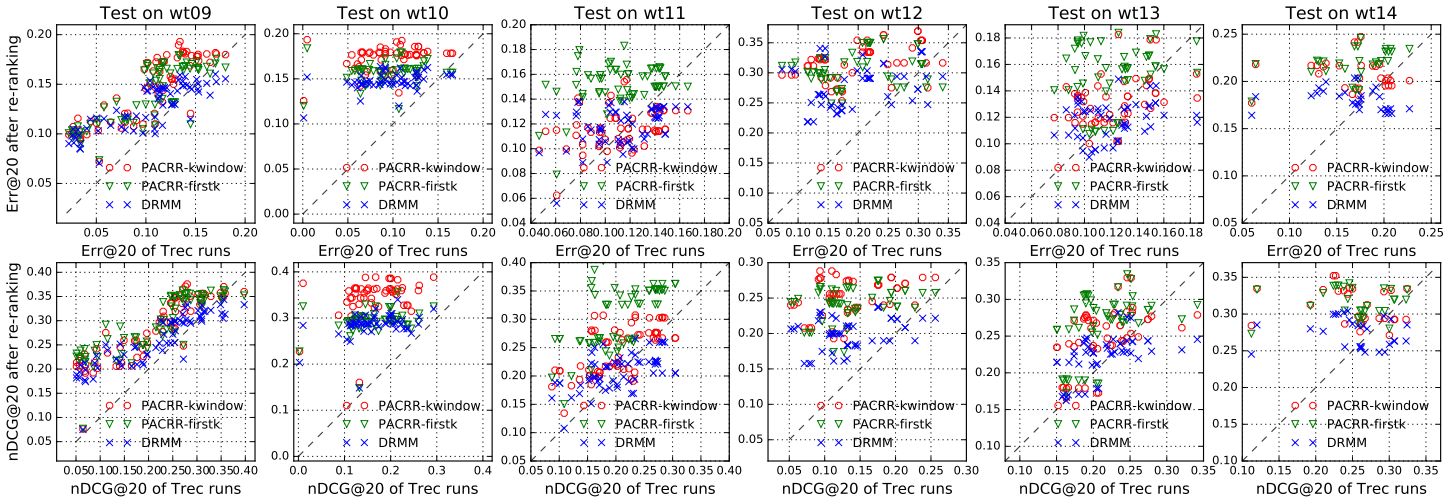

Figure 3: Re-ranking search results from Trec Web Track 2009–14 runs with *PACRR-kwindow*, *PACRR-firstk* and *DRMM*, respectively. The x-axis represents the ranking score of the original Trec runs and the y-axis represents the score of the corresponding re-ranked runs. The dashed line is y=x. Err@20 is used in the upper row and nDCG@20 is used in the lower row.

Tested Methods	Metrics	year	wt09	wt10	wt11	wt12	wt13	wt14
		# total runs	71	55	62	50	52	29
average Δ metric score over each year (%): $\frac{\text{re-rank score}-\text{original score}}{\text{original score}}$	Err@20	*DRMM*	54%	315%	11%	61%	5%	19%
		PACRR-firstk	66% †	362%	43% †	76% †	37% †	41% †
		PACRR-kwindow	70% †	393% †	10%	83% †	21% †	36% †
	nDCG@20	*DRMM*	49%	274%	8%	70%	9%	15%
		PACRR-firstk	69% †	304% †	56% †	100% †	31% †	31% †
		PACRR-kwindow	63% †	345% †	27% †	113% †	23% †	30% †
# runs get higher metric score after re-ranking	Err@20	*DRMM*	58	52	29	43	21	19
		PACRR-firstk	67	52	**60**	46	**45**	**29**
		PACRR-kwindow	**69**	**55**	29	**48**	34	22
	nDCG@20	*DRMM*	61	**55**	31	44	32	16
		PACRR-firstk	**67**	**55**	**62**	**50**	**48**	**27**
		PACRR-kwindow	66	**55**	52	**50**	42	25

Table 2: The average statistics when re-ranking all runs from the Trec Web Track 2009–14 based on Err@20 and nDCG@20. The average differences of the metric score for individual runs are reported in the 1st part. Two-tailed paired Student's t-test with 95% confidence level are computed over the differences among different runs, and † indicates significance relative to the number for *DRMM*. The number of runs that show improvements in terms of a metric is summarized at the bottom.

re-ranking. Therefore, the points in the upper left correspond to the ones showing improved performance when re-ranking, and vice versa for the lower right region. In Table 2, we provide statistics for the relative changes of metrics before and after re-ranking and report them in terms of percentages ("average Δ metric score"). The number of systems whose results on individual years has increased after re-ranking with a tested model is reported in the bottom part of Table 2. Note that these results assess two different aspects: the average Δ metric score in Table 2 captures the degree to which a model can improve an initial run, while the number of runs indicate to what extent an improvement can be achieved across different systems. In other words, the former measures the strength of the models, and the latter demonstrates the adaptability of the models.

We first consider the bottom left three plots in Figure 3, corresponding to results on 2009–11 under nDCG@20, similar to the ones reported in [8]. It can be observed that *DRMM* fares well. For example, the nDCG@20 of *QL* from [8] is 0.224 (cf. Table 2 therein), and the nDCG@20 after re-ranking is 0.258, while in Figure 3, the re-ranked runs with initial nDCG@20 between 0.2 and 0.25 have nDCG@20 around 0.3, which is especially true on 2011 and 2012. Given that *DRMM* outperforms *DSSM* [10] and *C-DSSM* [21] on the same group of test queries in [8], we argue that the *DRMM* trained in our work is a very strong baseline.

Both Table 2 and Figure 3 confirm that either *PACRR* variant can improve by at least 23% on average across different years in terms of nDCG@20. Beyond that, 80% of submitted runs are improved after re-ranking by the proposed models. For 2010–12, all submitted runs are consistently improved by *PACRR-firstk* in terms of nDCG@20. Both variants of *PACRR* can significantly outperform *DRMM* on all six years in terms of average improvements, where the statistical tests are conducted between the improvements over individual runs when re-ranked by different models. In addition, Figure 3 provides a more direct way to compare the tested models. When measuring Err@20, in all plots the markers corresponding to *PACRR-kwindow* or *PACRR-firstk* are largely located above the blue crosses. Meanwhile, comparing the two model variants, *PACRR-firstk* obviously performs better on 2011 and 2013, while *PACRR-kwindow* fares better on 2010. For the other years, the distinction is less conclusive, as the markers for the two approaches are mixed together.

4.4 Ranking Document Pairs

As pointed out in [18], the ranking of documents can be decomposed into rankings of document pairs. In other words, a model's retrieval quality can be examined by checking across a range of individual document pairs how likely it is that the model assigns a higher score for documents that are more relevant. Thus, it is possible for us to compare different models over the same set of complete judgments, removing the issue of different initial runs. Moreover, although ranking is our ultimate task, a direct inspection of pairwise prediction results can indicate which kinds of document pairs a model succeeds at or fails on. We first convert the graded judgments from Trec into ranked document pairs by comparing their labels. In particular, document pairs are created among documents that have different labels. A prediction is counted as correct if it assigns a higher score to the document from the pair that is labeled with a higher degree of relevance. The judgments from Trec contain at most six relevance levels: junk pages (*Junk*), non-relevance (*NRel*), relevance (*Rel*), high relevance (*HRel*), key pages (*Key*) and navigational pages (*Nav*), corresponding to six graded levels, i.e., -2, 0, 1, 2, 3, 4. Different years have different levels and different grade assignment policies, so we merge and unify the original levels from the six years into four grades: *Nav*, *HRel*, *Rel* and *NRel*. Given the absence of *HRel* and *Nav* on some queries, we first summarize the results by macro-averaging over each year in Figure 4. Specifically, we divide the number of correctly predicted document pairs by the total number of documents from a year (for 50 queries). This captures the average accuracy as a bar for each label combination. The statistics are calculated over all possible combinations of judgment pairs. We further group accuracy number according to their type of label combination over individual queries. The statistics are summarized in Table 3. The column named volume is the percentage of the current label combination out of all document pairs, and the # query column statistics the number of queries where there exists a label combination. In Table 3, we observe that *PACRR* model variants always perform significantly better on label combinations *HRel* vs. *NRel*, *Rel* vs. *NRel* and *Nav* vs. *NRel*, which in total cover 90% of all document pairs. Meanwhile, apart from *Nav*-*NRel*, there is no significant difference when distinguishing *Nav* from other types.

4.5 Discussion

Hyper-parameters. As mentioned in Section 4.1, models are selected based on the Err@20 over validation data. Hence, it is sufficient to to use a reasonable and representative validation dataset, rather than handpicking a specific set of parameter settings. However, to have a better understanding of the influence of different hyper-parameters, we explore *PACRR-kwindow*'s effectiveness when several hyper-parameters are varied. The results when re-ranking *QL-ALL* search results are given in Figure 5. The results are reported based on the models with the highest validation scores after fixing certain hyper-parameters. For example, the Err@20 in the leftmost figure is obtained when fixing l_d to the values shown. The crosses in Figure 5 correspond to the models that were selected for use on the test data, based on their validation set scores. It can be seen that the selected models are not necessarily the best model on the test data, as evidenced by the differences between validation and test data results, but we consistently obtain scores within a reasonable margin. Owing to space considerations, we omit the plots for *PACRR-firstk*.

Choice between *kwindow* and *firstk* approaches. As mentioned, both *PACRR-kwindow* and *PACRR-firstk* serve to address the variable-length challenge for documents and queries, and to make the training feasible and more efficient. Our experiments demonstrate that both distillation methods have comparable performance in that both significantly outperform the *DRMM* and *QL* baselines. In general, if both training and test documents are known to be short enough to fit in memory, then *PACRR-firstk* can be used directly. Otherwise, *PACRR-kwindow* is a reasonable choice to provide comparable results. Alternatively, one can regard this choice as another hyper-parameter, and make a selection based on held-out validation data.

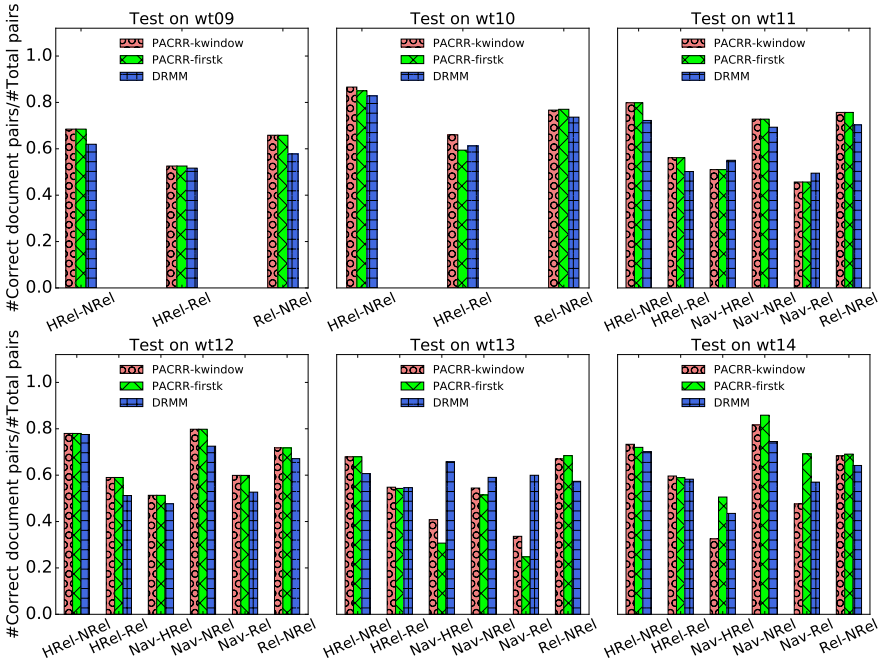

Figure 4: Comparison of *PACRR-kwindow*, *PACRR-firstk* and *DRMM* over document pairs that are ranked in agreement with the judgments from Trec. The x-axis is the pair of labels from Trec judgments. The y-axis is the the ratio between the number of correctly ranked document pairs and the total number. All judgments from Trec are considered.

label combinations	volume (%)	# queries	tested methods		
			PACRR-kwindow	*PACRR-firstk*	*DRMM*
Nav-HRel	0.3%	49	45.5%	45.8%	48.2%
Nav-Rel	1.1%	65	56.3%	56.0%	57%
Nav-NRel	3.6%	67	76.6% †	76.1% †	71.5%
HRel-Rel	8.4%	257	57.0%	57.3%	55.8%
HRel-NRel	23.1%	262	76.4% †	76.7% †	70.1%
Rel-NRel	63.5%	290	72.5% †	73.0% †	65.9%
weighted average			72.0%	72.4%	66.1%

Table 3: Comparison among tested methods in terms of accuracy in ranking document pairs with different labels. The column named "volume" records the occurrences of each label combination out of the total pairs. The # queries column records the number of queries that include a particular label combination. † indicates significant difference relative to the accuracy of *DRMM*, when using two-tailed paired Student's t-test at a 95% confidence level. In the last row, the average accuracy among different kinds of label combinations is computed, weighted by their corresponding volume.

Accuracy in sorting document pairs. Beyond the observations from Section 4.4, we further examine the methods' accuracy over binary judgments by merging the *Nav*, *HRel* and *Rel* labels. The accuracies become 73.5%, 74.1% and 67.4% for *PACRR-kwindow*, *PACRR-firstk*, and *DRMM*, respectively. Note that the manual judgments that indicate a document as relevant or non-relevant relative to a given query includes disagreements [3, 23] and errors [2]. In particular, a 64% agreement (cf. Table 2(b) therein) is observed over the inferred relative order among document pairs based on graded judgments from six trained judges [3]. When reproducing Trec judgments, AI-Maskari et al. [1] reported a 74% agreement (cf. Table 1 therein) with the original judgments from Trec when a group of users re-judged 56 queries on the Trec-8 document collections. Meanwhile, Alonso and Mizzaro [2] observed a 77% agreement relative to judgments from Trec when collecting judgments via crowdsourcing. Therefore, the 74% agreement achieved by both *PACRR* methods is close to the aforementioned agreements among different human assessors. However, when distinguishing *Nav*, *HRel* and *Rel*, the tested models still fall significantly short of human-level results. These distinctions are important for a successful ranker, especially when measuring with graded metrics such as Err@20 and nDCG@20. Hence, further research is needed for better discrimination among relevant documents with different degrees of relevance. In addition, as for the distinction between *Nav*

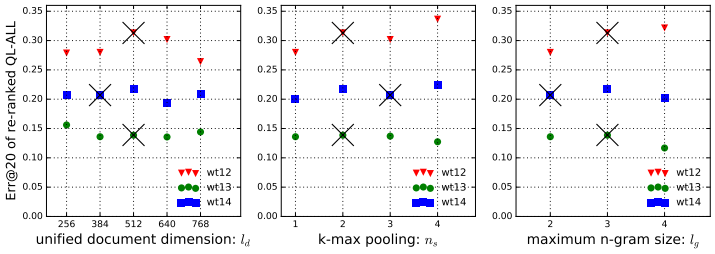

Figure 5: The Err@20 of re-ranked *QL-ALL* with *PACRR-kwindow* when applying different hyper-parameters: l_d, n_s and l_g. The x-axis reflects the parameter settings for different hyper-parameters, and the y-axis the Err@20. Points covered with crosses correspond to the ones reported in Table 1.

documents and *Rel* or *HRel* documents, we argue that since *Nav* actually indicates that a document mainly satisfies a navigational intent, this makes such documents qualitatively different from *Rel* and *HRel* documents. Specifically, a *Nav* is more relevant for a user with navigational intent, whereas for other users it may in some cases be less useful than a document that directly includes highly pertinent information content. Therefore, we hypothesize that further improvements can be obtained by introducing a classifier for navigational documents before employing our proposed deep retrieval model.

5 CONCLUSION

In this work, we have highlighted and demonstrated the importance of preserving positional information for deep retrieval models by incorporating insights from traditional information retrieval models into our *PACRR* novel deep retrieval model. In particular, our model captures term dependencies and proximity among query term occurrences through the use of convolutional layers that consider document and query n-grams of different sizes. Our model considers document relevance across different query terms through the use of a recurrent layer that combines relevance signals across query terms. Extensive experiments on Trec Web Track ad-hoc data show that the proposed model both substantially outperforms the state-of-the-art deep model on these data and can dramatically improve search results as a re-ranker.

REFERENCES

[1] Azzah Al-Maskari, Mark Sanderson, and Paul Clough. 2008. Relevance judgments between TREC and Non-TREC assessors. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 683–684.

[2] Omar Alonso and Stefano Mizzaro. 2012. Using crowdsourcing for TREC relevance assessment. *Information Processing & Management* 48, 6 (2012), 1053–1066.

[3] Ben Carterette, Paul N Bennett, David Maxwell Chickering, and Susan T Dumais. 2008. Here or there: Preference Judgments for Relevance. In *Advances in Information Retrieval*. Springer, 16–27.

[4] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. 2009. Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM conference on Information and knowledge management (CIKM '09)*. ACM, New York, NY, USA, 621–630. DOI: http://dx.doi.org/10.1145/1645953.1646033

[5] Daniel Cohen, Qingyao Ai, and W. Bruce Croft. 2016. Adaptability of Neural Networks on Varying Granularity IR Tasks. *CoRR* abs/1606.07565 (2016). http://arxiv.org/abs/1606.07565

[6] Kevyn Collins-Thompson, Craig Macdonald, Paul Bennett, Fernando Diaz, and Ellen M Voorhees. 2015. *TREC 2014 web track overview*. Technical Report. DTIC Document.

[7] Stephen Cronen-Townsend, Yun Zhou, and W. Bruce Croft. 2004. A framework for selective query expansion. In *Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004*, David A. Grossman, Luis Gravano, ChengXiang Zhai, Otthein Herzog, and David A. Evans (Eds.). ACM, 236–237. DOI: http://dx.doi.org/10.1145/1031171.1031220

[8] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. 2016. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 55–64.

[9] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (Nov. 1997), 1735–1780. DOI: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[10] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management (CIKM '13)*. ACM, New York, NY, USA, 2333–2338. DOI: http://dx.doi.org/10.1145/2505515.2505665

[11] Samuel Huston and W. Bruce Croft. 2014. A Comparison of Retrieval Models using Term Dependencies. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, Jianzhong Li, Xiaoyang Sean Wang, Minos N. Garofalakis, Ian Soboroff, Torsten Suel, and Min Wang (Eds.). ACM, 111–120. DOI: http://dx.doi.org/10.1145/2661829.2661894

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

[13] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188* (2014).

[14] Donald Metzler and W Bruce Croft. 2005. A Markov random field model for term dependencies. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 472–479.

[15] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2016. Learning to Match Using Local and Distributed Representations of Text for Web Search. *arXiv preprint arXiv:1610.08136* (2016).

[16] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, and Xueqi Cheng. 2016. A Study of MatchPyramid Models on Ad-hoc Retrieval. *CoRR* abs/1606.04648 (2016). http://arxiv.org/abs/1606.04648

[17] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2016. Text Matching As Image Recognition. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16)*. 2793–2799.

[18] Kira Radinsky and Nir Ailon. 2011. Ranking from Pairs and Triplets: Information Quality, Evaluation Methods and Query Complexity. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining (WSDM '11)*. ACM, New York, NY, USA, 105–114. DOI: http://dx.doi.org/10.1145/1935826.1935850

[19] Aliaksei Severyn and Alessandro Moschitti. 2015. Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '15)*.

[20] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM, 101–110.

[21] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning Semantic Representations Using Convolutional Neural Networks for Web Search. In *Proceedings of the 23rd International Conference on World Wide Web (WWW '14 Companion)*.

[22] Tao Tao and ChengXiang Zhai. 2007. An exploration of proximity measures in information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 295–302.

[23] Ellen M Voorhees. 2000. Variations in relevance judgments and the measurement of retrieval effectiveness. *Information processing & management* 36, 5 (2000), 697–716.

[24] Jinxi Xu and W. Bruce Croft. 1996. Query Expansion Using Local and Global Document Analysis. In *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)*, Hans-Peter Frei, Donna Harman, Peter Schäuble, and Ross Wilkinson (Eds.). ACM, 4–11. DOI: http://dx.doi.org/10.1145/243199.243202

[25] Chengxiang Zhai and John Lafferty. 2001. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 334–342.