PACRR: A Position-Aware Neural IR Model for Relevance Matching

Kai Hui
MPI for Informatics
Saarbrücken
Graduate School of
Computer Science

Andrew Yates
MPI for Informatics

Klaus Berberich
htw saar
MPI for Informatics

Gerard de Melo
Rutgers University
New Brunswick

Abstract

In order to adopt deep learning for information retrieval, models are needed that can capture all relevant information required to assess the relevance of a document to a given user query. While previous works have successfully captured unigram term matches, how to fully employ position-dependent information such as proximity and term dependencies has been insufficiently explored. In this work, we propose a novel neural IR model named *PACRR* aiming at better modeling position-dependent interactions between a query and a document. Extensive experiments on six years' TREC Web Track data confirm that the proposed model yields better results under multiple benchmarks.

1 Introduction

Despite the widespread use of deep neural models across a range of linguistic tasks, to what extent such models can improve information retrieval (IR) and which components a deep neural model for IR should include remain open questions. In ad-hoc IR, the goal is to produce a ranking of relevant documents given an open-domain ("ad hoc") query and a document collection. A ranking model thus aims at evaluating the interactions between different documents and a query, assigning higher scores to documents that better match the query. Learning to rank models, like the recent IRGAN model (Wang et al., 2017), rely on handcrafted features to encode query document interactions, e.g., the relevance scores from unsupervised ranking models. Neural IR models differ in that they extract interactions directly based on the queries and documents. Many early neural IR models can be categorized as *semantic matching* models, as they embed both queries and documents into a low-dimensional space, and then assess their similarity based on such dense representations. Examples in this regard include *DSSM* (Huang et al., 2013) and *DESM* (Mitra et al., 2016). The notion of relevance is inherently asymmetric, however, making it different from well-studied semantic matching tasks such as semantic relatedness and paraphrase detection. Instead, *relevance matching* models such as MatchPyramid (Pang et al., 2016), DRMM (Guo et al., 2016) and the recent K-NRM (Xiong et al., 2017) resemble traditional IR retrieval measures in that they directly consider the relevance of documents' contents with respect to the query. The DUET model (Mitra et al., 2017) is a hybrid approach that combines signals from a local model for relevance matching and a distributed model for semantic matching. The two classes of models are fairly distinct. In this work, we focus on relevance matching models.

Given that relevance matching approaches mirror ideas from traditional retrieval models, the decades of research on ad-hoc IR can guide us with regard to the specific kinds of relevance signals a model ought to capture. Unigram matches are the most obvious signals to be modeled, as a counterpart to the term frequencies that appear in almost all traditional retrieval models. Beyond this, positional information, including where query terms occur and how they depend on each other, can also be exploited, as demonstrated in retrieval models that are aware of term proximity (Tao and Zhai, 2007) and term dependencies (Huston and Croft, 2014; Metzler and Croft, 2005). Query coverage is another factor that can be used to ensure that, for queries with multiple terms, top-ranked documents contain multiple query terms rather than emphasizing only one query term. For example, given the query "dog adoption requirements", unigram matching signals correspond to the occurrences of the individual terms "dog", "adoption", or "requirements". When considering positional information, text passages with "dog adoption" or "requirements for dog adoption" are highlighted, distinguishing them from text that only includes individual terms. Query coverage, meanwhile, further emphasizes that matching signals for "dog", "adoption", and "requirements" should all be included in a document.

Similarity signals from unigram matches are taken as input by DRMM (Guo et al., 2016) after being summarized as histograms, whereas K-NRM (Xiong et al., 2017) directly digests a query-document similarity matrix and summarizes it with multiple kernel functions. As for positional information, both the MatchPyramid (Pang et al., 2016) and local DUET (Mitra et al., 2017) models account for it by incorporating convolutional layers based on similarity matrices between queries and documents. Although this leads to more complex models, both have difficulty in significantly outperforming the DRMM model (Guo et al., 2016; Mitra et al., 2017). This indicates that it is non-trivial to go beyond unigrams by utilizing positional information in deep neural IR models. Intuitively, unlike in standard sequence-based models, the interactions between a query and a document are sequential along the query axis as well as along the document axis, making the problem multi-dimensional in nature. In addition, this makes it non-trivial to combine matching signals from different parts of the documents and over different query terms. In fact, we argue that both MatchPyramid and local DUET models fail to fully account for one or more of the aforementioned factors. For example, as a pioneering work, MatchPyramid is mainly motivated by models developed in computer vision, resulting in its disregard of certain IR-specific considerations in the design of components, such as pooling sizes that ignore the query and document dimensions. Meanwhile, local DUET's CNN filters match entire documents against individual query terms, neglecting proximity and possible dependencies among different query terms.

We conjecture that a suitable combination of convolutional kernels and recurrent layers can lead to a model that better accounts for these factors. In particular, we present a novel re-ranking model called *PACRR* (*Position-Aware Convolutional-Recurrent Relevance* Matching). Our approach first produces similarity matrices that record the semantic similarity between each query term and each individual term occurring in a document. These matrices are then fed through a series of convolutional, max-k-pooling, and recurrent layers so as to capture interactions corresponding to, for instance, bigram and trigram matches, and finally to aggregate the signals in order to produce global relevance assessments. In our model, the convolutional layers are designed to capture both unigram matching and positional information over text windows with different lengths; k-max pooling layers are along the query dimension, preserving matching signals over different query terms; the recurrent layer combines signals from different query terms to produce a query-document relevance score.

Organization. The rest of this paper unfolds as follows. Section 2 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 3, before concluding in Section 4.

2 The PACRR Model

We now describe our proposed *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 $rel(q,d)$. Note that in principle the proposed model can be trained end-to-end by backpropagating through the word embeddings, as in (Xiong et al., 2017). In this work, however, we focus on highlighting the building blocks aiming at capturing positional information, and freeze the word embedding layer to achieve better efficiency. The pipeline is summarized in Figure 1.

2.1 Relevance Matching

We first encode the query-document relevance matching via query-document similarity matrices $sim_{|q|\times|d|}$ that encodes the similarity between terms from a query q and a document d, where sim_{ij} corresponds to the similarity between the i-th term from q and the j-th term from d. When using cosine similarity, we have

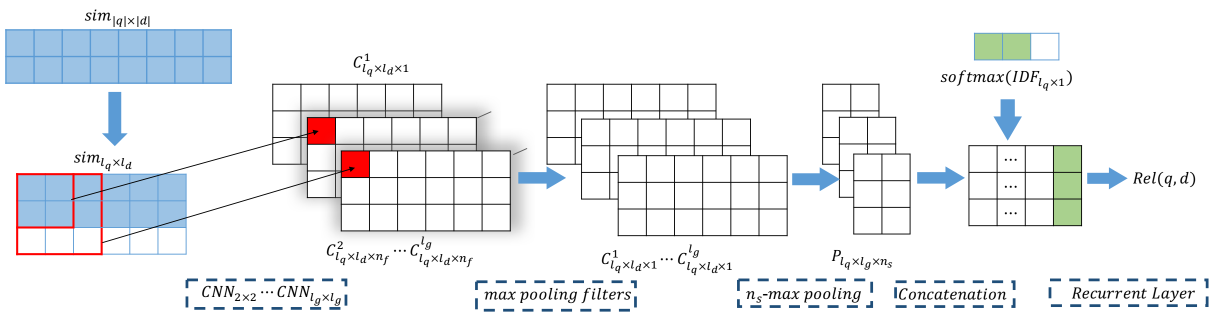
$sim_{|q|\times|d|}$
$sim_{l_q\times l_d}$
$C^1_{l_q\times l_d\times 1}$
$C^2_{l_q\times l_d\times n_f}\cdots C^{l_g}_{l_q\times l_d\times n_f}$
$C^1_{l_q\times l_d\times 1}\cdots C^{l_g}_{l_q\times l_d\times 1}$
$P_{l_q\times l_g\times n_s}$
$softmax(IDF_{l_q\times 1})$
$Rel(q,d)$
$CNN_{2\times 2}\cdots CNN_{l_g\times l_g}$
max pooling filters
n_s-max pooling
Concatenation
Recurrent Layer

Figure 1: The pipeline of *PACRR*. Each query q and document d is first converted into a query-document similarity matrix $sim_{|q|\times|d|}$. Thereafter, a distillation method (*firstk* is displayed) transforms the raw similarity matrix into unified dimensions, namely, $sim_{l_q\times l_d}$. Here, $l_g - 1$ convolutional layers (CNN) are applied to the distilled similarity matrices. As $l_g = 3$ is shown, layers with kernel size 2 and 3 are applied. Next, max pooling is applied, leading to 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 for $n_s = 2$ is shown here. Finally, the similarity signals from different n-gram sizes are concatenated, the query terms' normalized IDFs are added, and a recurrent layer combines these signals for each query term into a query-document relevance score $rel(q,d)$.

$sim \in [-1,1]^{|q|\times|d|}$. As suggested in (Hui et al., 2017), query-document similarity matrices preserve a rich signal that can be used to perform relevance matching beyond unigram matches. In particular, n-gram matching corresponds to consecutive document terms that are highly similar to at least one of the query terms. Query coverage is reflected in the number of rows in *sim* that include at least one cell with high similarity. The similarity between a query term q and document term d is calculated by taking the cosine similarity using the pre-trained[1] *word2vec* (Mikolov et al., 2013).

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. Akin to (Mitra et al., 2017), 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.

PACRR-kwindow. As suggested in (Guo et al., 2016), relevance matching is local. Document terms that have a low query similarity relative to a document's other terms cannot contribute substantially to the document's relevance score. Relevance matching can be extracted in terms of pieces of text that include relevant information. That is, one can segment documents according to relevance relative to the given query and retain only 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 for *text snippets* beyond length n, we produce a similarity matrix $sim^n_{l_q\times l_d}$ for each query-document pair and each n, because n consecutive terms must be co-considered later on. For each text snippet with length n 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 averaging 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) since it can consider at most n consecutive terms that are

[1]https://code.google.com/archive/p/word2vec/

present in the original document. This variant's output is a similarity matrix $sim^n_{l_q \times l_d}$ for each n.

2.2 Deep Retrieval Model

Given a query-document similarity matrix $sim_{l_q \times l_d}$ as input, our deep architecture relies on convolutional layers to match every text snippet with length n in a query and in a document to produce similarity signals for different n. Subsequently, two consecutive max pooling layers extract the document's strongest similarity cues for each n. Finally, a recurrent layer aggregates these salient signals to predict a global query-document relevance score $rel(q, d)$.

Convolutional relevance matching over local text snippets. The purpose of this step is to match text snippets with different length from a query and a document given their 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; 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. 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 with a $n \times n$ kernel. 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 length of the longest text snippet 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 matching signals on n-term text snippets. 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 $\mathcal{C}^n_{l_q \times l_d \times n_f}$, and the original similarity matrix is directly employed to handle the signals over unigrams.

Two max pooling layers. The purpose of this step is to capture the n_s strongest similarity signals for each query term. Measuring the similarity signals 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 patterns, 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 opportunity 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, assuming that there only exists one particular true matching pattern in a given $n \times n$ window, which serves different purposes compared with other tasks, such as the sub-sampling in computer vision. We then perform k-max pooling (Kalchbrenner et al., 2014) over the query dimension l_q 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 $\mathcal{C}^i$ from the convolutional layer and on the original similarity matrix, which captures unigram matching, to produce the 3-dimensional tensor $\mathcal{P}_{l_q \times l_g \times n_s}$. This tensor contains the n_s strongest 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 as input and learning weights to transform them into the final relevance score. More precisely, akin to (Guo et al., 2016), the IDF of each query term q_i is passed through a softmax layer for normalization. Thereafter, we split up the query term dimension to produce a matrix $\mathcal{P}_{l_g \times n_s}$ for each query term q_i, subsequently forming 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 (Hochreiter and Schmidhuber, 1997) 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 aforementioned salient signals for the query term along different kernel sizes. The LSTM's output is then used as our document relevance score $rel(q, d)$.

Training objective. Our model is trained on triples consisting of a query q, relevant document d^+, and non-relevant document d^-, minimizing a standard pairwise max margin loss as in Eq. 1.

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

3 Evaluation

In this section, we empirically evaluate *PACRR* models using manual relevance judgments from the standard TREC Web Track. We compare them against several state-of-the-art neural IR models[2], including *DRMM* (Guo et al., 2016), DUET (Mitra et al., 2017), *MatchPyramid* (Pang et al., 2016), and *K-NRM* (Xiong et al., 2017). The comparisons are over three task settings: re-ranking search results from a simple initial ranker (RERANKSIMPLE); re-ranking all runs from the TREC Web Track (RERANKALL); and examining neural IR models' classification accuracy between document pairs (PAIRACCURACY).

3.1 Experimental Setup

We rely on the widely-used 2009–2014 TREC Web Track ad-hoc task benchmarks[3]. The benchmarks are based on the CLUEWEB09 and CLUEWEB12 datasets as document collections. In total, there are 300 queries and more than 100k judgments (qrels). Three years (2012–14) of query-likelihood baselines[4] provided by TREC[5] serve as baseline runs in the RERANKSIMPLE benchmark. In the RERANKALL setting, the search results from runs submitted by participants from each year are also considered: there are 71 (2009), 55 (2010), 62 (2011), 48 (2012), 50 (2013), and 27 (2014) runs. ERR@20 (Chapelle et al., 2009) and nDCG@20 (Järvelin and Kekäläinen, 2002) are employed as evaluation measures, and both are computed with the script from TREC[6].

Training. At each step, we perform Stochastic Gradient Descent (SGD) with 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*[7] 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

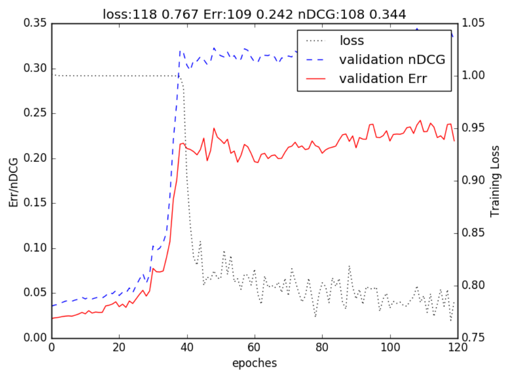

Figure 2: The training loss, ERR@20 and nDCG@20 per iteration on validation data when training on Web Track 2010–14. The x-axis denotes the iterations. The y-axis indicates the ERR@20/nDCG@20 (left) and the loss (right). The best performance appears on 109th iteration with ERR@20=0.242. The lowest training loss (0.767) occurs after 118 iterations.

[2]We also attempted to include IRGAN (Wang et al., 2017) model as a baseline, but failed to obtain reasonable results when training on TREC data.

[3]`http://trec.nist.gov/tracks.html`

[4]Terrier (Ounis et al., 2006) version without filtering spam documents

[5]`https://github.com/trec-web/trec-web-2014`

[6]http://trec.nist.gov/data/web/12/gdeval.pl

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

given number of iterations is reached. The model is saved at every iteration. We use the model with the best ERR@20 on the validation set to make predictions. Proceeding in a round-robin manner, 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, model parameters and training iterations are chosen by maximizing the ERR@20 on the validation set. The selected model is then used to make predictions on the test data. An example of this training procedure is shown in Figure 2. There are four hyper-parameters that govern the behavior of the proposed *PACRR-kwindow* and *PACRR-firstk*: the unified length of 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 consider based on pilot experiments and domain insights. In particular, we evaluate $l_d \in [256, 384, 512, 640, 768]$, $n_s \in [1, 2, 3, 4]$, and $l_g \in [2, 3, 4]$. Due to the limited possible matching patterns given a small kernel size (e.g., $l_g = 3$), n_f is fixed to 32. For *PACRR-firstk*, we intuitively desire to retain as much information as possible from the input, and thus l_d is always set to 768.

DRMM (*DRMM*$_{LCH\times IDF}$), *DUET*, *MatchPyramid* and *K-NRM* are trained under the same settings using the hyperparameters described in their respective papers. In particular, as our focus is on the deep relevance matching model as mentioned in Section 1, we only compare against DUET's local model, denoted as *DUETL*. In addition, *K-NRM* is trained slightly different from the one described in (Xiong et al., 2017), namely, with a frozen word embedding layer. This is to guarantee its fair comparison with other models, given that most of the compared models can be enhanced by co-training the embedding layers, whereas the focus here is the strength coming from the model architecture. A fully connected middle layer with 30 neurons is added to compensate for the reduction of trainable parameters in *K-NRM*, mirroring the size of DRMM's first fully connected layer.

All models are implemented with Keras (Chollet et al., 2015) using Tensorflow as backend, and are trained on servers with multiple CPU cores. In particular, the training of *PACRR* takes 35 seconds per iteration on average, and in total at most 150 iterations are trained for each model variant.

3.2 Results

RERANKSIMPLE. We first examine the proposed model by re-ranking the search results from the *QL* baseline on Web Track 2012–14. The results are summarized in Table 1. It can be seen that *DRMM* can significantly improve *QL* on WT12 and WT14, whereas *MatchPyramid* fails on WT12 under ERR@20. While *DUETL* and *K-NRM* can consistently outperform *QL*, the two variants of *PACRR* are the only models that can achieve significant improvements at a 95% significance level on all years under both ERR@20 and nDCG@20. More remarkably, by solely re-ranking the search results from *QL*, *PACRR-firstk* can already rank within the top-3 participating systems on all three years as measured by both ERR and nDCG. The re-ranked search results from *PACRR-kwindow* also ranks within the top-5 based on nDCG@20. On average, both *PACRR-kwindow* and *PACRR-firstk* achieve 60% improvements over *QL*.

RERANKALL. In this part, we would like to further examine the performance of the proposed models in re-ranking different sets of search results. Thus, we extend our analysis to re-rank search results from all submitted runs from six years of the TREC Web Track ad-hoc task. In particular, we only consider the judged documents from TREC, which loosely correspond to top-20 documents in each run. 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 scores for each re-ranked run. Instead, we summarize the results by comparing the performance of each run before and after re-ranking, and provide statistics over each year to compare the methods under consideration in Table 2. In the top portion of Table 2, we report the relative changes in metrics before and after re-ranking in terms of percentages ("average Δ measure score"). In the bottom portion, we report the percentage of systems whose results have increased after re-ranking. Note that these results assess two different aspects: the average Δ measure score in Table 2 captures the degree to which a model can improve an initial run, while

Measure	Years	PACRR-firstk	Rank	PACRR-kwindow	Rank	DUETL	Rank	DRMM	Rank	MatchPyramid	Rank	K-NRM	Rank	QL	Rank
ERR@20	wt12	0.318 (mQ)	2	0.313 (MQ)	4	0.281 (Q)	10	0.289 (Q)	10	0.227	16	0.258 (Q)	12	0.177	26
	wt13	0.166 (DKQ)	3	0.139 (Q)	14	0.147 (Q)	12	0.124	25	0.141 (q)	13	0.134 (q)	14	0.101	38
	wt14	0.221 (LMQ)	2	0.208 (Q)	3	0.179 (Q)	12	0.193 (Q)	10	0.176 (Q)	12	0.201 (Q)	8	0.131	25
nDCG@20	wt12	0.243 ($DLMQ$)	2	0.250 ($DLMQ$)	2	0.186 (Q)	11	0.197 (Q)	8	0.164 (Q)	16	0.222 (Q)	4	0.106	39
	wt13	0.295 ($DLkQ$)	3	0.279 (DQ)	4	0.248 (q)	11	0.228	20	0.258 (Q)	7	0.251 (Q)	11	0.190	36
	wt14	0.339 (LMQ)	1	0.331 (LMQ)	1	0.267 (q)	11	0.300 (Q)	6	0.278 (Q)	10	0.324 (Q)	2	0.231	23

Table 1: ERR@20 and nDCG@20 on TREC Web Track 2012–14 when re-ranking search results from *QL*. The comparisons are conducted between two variants of PACRR and DRMM (D/d), DUETL (L/l), MatchPyramid (M/m) and K-NRM (K/k). All methods are compared against the *QL* (Q/q) baseline. The upper/lower-case characters in the brackets indicate a significant difference under two-tailed paired Student's t-tests at 95% or 90% confidence levels relative to the corresponding approach. In addition, the relative ranks among all runs within the respective years according to ERR@20 and nDCG@20 are also reported directly after the absolute scores.

	Measures	Tested Methods	wt09	wt10	wt11	wt12	wt13	wt14
average Δ measure score over each year (%): $\frac{\text{re-rank score}-\text{original score}}{\text{original score}}$	ERR@20	*PACRR-firstk*	66% (DLK)	362% (dm)	43% ($DLMK$)	76% ($DLMK$)	37% ($DLMK$)	41% ($DLMK$)
		PACRR-kwindow	70% ($DLmK$)	393% (DlM)	10% (LMK)	83% ($DLMK$)	21% (DLM)	36% ($DLMK$)
		DUETL	80% (DMK)	316%	15% (DMK)	64% (M)	26% (DM)	19% (MK)
		DRMM	54% (LMK)	315%	11% (LMK)	61% (M)	5% (LMK)	19% (MK)
		MatchPyramid	65% (DL)	313%	2% (DLK)	48% (DLK)	29% (DLK)	14% (DLK)
		K-NRM	59% (DL)	333%	31% (DLM)	63% (M)	25% (DM)	32% (DLM)
	nDCG@20	*PACRR-firstk*	69% ($DLMK$)	304% (LM)	56% ($DLMK$)	100% ($DLMK$)	31% ($DLMK$)	31% (DLM)
		PACRR-kwindow	63% (DmK)	345% ($DLMK$)	27% ($DLMK$)	113% ($DLMK$)	23% (DLK)	30% (DLM)
		DUETL	62% (DMK)	237% (DK)	17% (DMK)	55% (DMK)	17% (DMK)	10% (DMK)
		DRMM	49% (LMK)	274% (LMk)	8% (LMK)	70% (LMK)	9% (LMK)	15% (LK)
		MatchPyramid	59% (DLk)	232% (DK)	1% (DLK)	37% (DLK)	21% (DLk)	14% (LK)
		K-NRM	52% (DLm)	288% (dLM)	36% (DLM)	85% (DLM)	19% (DLm)	30% (DLM)
% of runs that get better performance after re-ranking	ERR@20	*PACRR-firstk*	94%	95%	**97%**	92%	**87%**	**100%**
		PACRR-kwindow	**97%**	**100%**	47%	**96%**	65%	76%
		DUETL	94%	95%	61%	86%	69%	59%
		DRMM	82%	95%	47%	86%	40%	66%
		MatchPyramid	85%	93%	40%	78%	81%	59%
		K-NRM	87%	95%	89%	82%	67%	86%
	nDCG@20	*PACRR-firstk*	**94%**	**100%**	**100%**	**100%**	**92%**	**93%**
		PACRR-kwindow	93%	**100%**	84%	**100%**	81%	86%
		DUETL	86%	93%	69%	92%	79%	59%
		DRMM	86%	**100%**	50%	88%	62%	55%
		MatchPyramid	76%	93%	39%	80%	81%	69%
		K-NRM	**94%**	**100%**	97%	96%	81%	**93%**

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 scores for individual runs are reported in the top portion. The comparisons are conducted between two variants of PACRR and DRMM (D/d), DUETL (L/l), MatchPyramid (M/m) and K-NRM (K/k). The upper/lower-case characters in parentheses indicate a significant difference under two-tailed paired Student's t-tests at 95% or 90% confidence levels, respectively, relative to the corresponding approach. The percentage of runs that show improvements in terms of a measure is summarized in the bottom portion.

the percentages of runs indicate to what extent an improvement can be achieved over runs from different systems. In other words, the former measures the strength of the models, while the latter measures the adaptability of the models. Both *PACRR* variants improve upon existing rankings by at least 10% across different years. Remarkably, in terms of nDCG@20, at least 80% of the submitted runs are improved after re-ranking by the proposed models on individual years, and on 2010–12, all submitted runs are consistently improved by *PACRR-firstk*. Moreover, both variants of *PACRR* can significantly outperform all baseline models on at least three years out of the six years in terms of average improvement. However, it is clear that none of the tested models can make consistent improvements over all submitted runs across all six years. In other words, there still exist document pairs that are predicted contradicting to the judgments from TREC. Thus, in the next part, we further investigate the performance in terms of prediction over document pairs.

Label Pairs	Volume (%)	# Queries	Tested Methods					
			PACRR-firstk	*PACRR-kwindow*	*DUETL*	*DRMM*	*MatchPyramid*	*K-NRM*
Nav-HRel	0.3%	49	45.8%	45.5%	45.2%	48.2%	47.3%	51.6%
Nav-Rel	1.1%	65	56.0% (m)	56.3% (M)	54%	57% (M)	53.2% (D)	57.4%
Nav-NRel	3.6%	67	76.1% $(DLMK)$	76.6% $(DLMK)$	67.1% (M)	71.5% (M)	64.7% (DLK)	70.8% (M)
HRel-Rel	8.4%	257	57.3%	57.0%	55.5%	55.8%	52.8%	56.1%
HRel-NRel	23.1%	262	76.7% $(DLMK)$	76.4% $(DLMK)$	68.4% (K)	70.1% (MK)	65.6% (DK)	72.5% (DLM)
Rel-NRel	63.5%	290	73.0% $(DLMK)$	72.5% $(DLMK)$	63.9% (DMK)	65.9% (LMK)	61.4% (DLK)	68.7% (DLM)
weighted average			72.4%	72.0%	64.2%	66.1%	61.6%	68.4%

Table 3: Comparison among tested methods in terms of accuracy when comparing document pairs with different labels. The "volume" column indicates the percentage of occurrences of each label combination out of the total pairs. The "# Queries" column records the number of queries that include a particular label combination. The comparisons are conducted between two variants of PACRR and DRMM (D/d), DUETL (L/l), MatchPyramid (M/m) and K-NRM (K/k). The upper/lower-case characters in parentheses indicate a significant difference under two-tailed paired Student's t-tests at 95% or 90% confidence levels, respectively, relative to the corresponding approach. In the last row, the average accuracy among different kinds of label combinations is computed, weighted by their corresponding volume.

PAIRACCURACY. The ranking of documents can be decomposed into rankings of document pairs as suggested in (Radinsky and Ailon, 2011). Specifically, a model's retrieval quality can be examined by checking across a range of individual document pairs, namely, how likely a model can assign a higher score for a more relevant document. 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 target, 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. 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, and we merge and unify the original levels from the six years into four grades, namely, *Nav*, *HRel*, *Rel* and *NRel*. We compute the accuracy for each pair of labels. The statistics are summarized in Table 3. The volume column lists the percentage of a given label combination out of all document pairs, and the # query column provides the number of queries for which the label combination exists. In Table 3, we observe that both *PACRR* models always perform better than all baselines 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-Rel*, there is no significant difference when distinguishing *Nav* from other types. *K-NRM* and *DRMM* perform better than the other two baseline models.

3.3 Discussion

Hyper-parameters. As mentioned, models are selected based on the ERR@20 over validation data. Hence, it is sufficient to use a reasonable and representative validation dataset, rather than handpicking a specific set of parameter settings. However, to gain 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* search results are given in Figure 3. 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 3 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 constraints, 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 chal-

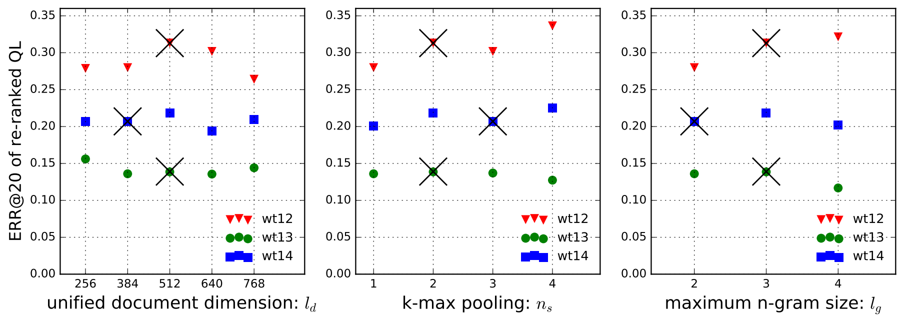

Figure 3: The ERR@20 of re-ranked *QL* with *PACRR-kwindow* when applying different hyper-parameters: l_d, n_s and l_g. The x-axis reflects the settings for hyper-parameters, and the y-axis is the ERR@20. Crosses correspond to the selected models.

lenge for documents and queries, and to make the training feasible and more efficient. 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.

Accuracy in PairAccuracy. Beyond the observations in Section 3.2, 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 contain disagreements (Carterette et al., 2008; Voorhees, 2000) and errors (Alonso and Mizzaro, 2012). 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 (Carterette et al., 2008). When reproducing TREC judgments, Al-Maskari et al. (Al-Maskari et al., 2008) 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 (Alonso and Mizzaro, 2012) observed a 77% agreement relative to judgments from TREC when collecting judgments via crowdsourcing. Therefore, the more than 73% agreement achieved by both *PACRR* methods is close to the aforementioned agreement levels among different human assessors. However, when distinguishing *Nav*, *HRel*, and *Rel*, the tested models still fall significantly short of the human judges' agreement levels. 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* 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 user intents, e.g., navigational pages, before employing neural IR models.

4 Conclusion

In this work, we have demonstrated the importance of preserving positional information for neural IR models by incorporating domain insights into the proposed *PACRR* model. In particular, *PACRR* captures term dependencies and proximity through multiple convolutional layers with different sizes. Thereafter, following two max-pooling layers, it combines salient signals over different query terms with a recurrent layer. Extensive experiments show that *PACRR* substantially outperforms four state-of-the-art neural IR models on TREC Web Track ad-hoc datasets and can dramatically improve search results when used as a re-ranking model.

References

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*, pages 683–684. ACM.

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

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*, pages 16–27. Springer.

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, pages 621–630, New York, NY, USA. ACM.

François Chollet et al. 2015. Keras. `https://github.com/fchollet/keras`.

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*, pages 55–64. ACM.

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

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, pages 2333–2338, New York, NY, USA. ACM.

Kai Hui, Andrew Yates, Klaus Berberich, and Gerard de Melo. 2017. Position-aware representations for relevance matching in neural information retrieval. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 799–800. International World Wide Web Conferences Steering Committee.

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*, pages 111–120. ACM.

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

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

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*, pages 472–479. ACM.

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

Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to match using local and distributed representations of text for web search. In *Proceedings of WWW 2017*. ACM.

Bhaskar Mitra, Eric Nalisnick, Nick Craswell, and Rich Caruana. 2016. A dual embedding space model for document ranking. *arXiv preprint arXiv:1602.01137*.

Iadh Ounis, Gianni Amati, Vassilis Plachouras, Ben He, Craig Macdonald, and Christina Lioma. 2006. Terrier: A high performance and scalable information retrieval platform. In *Proceedings of the OSIR Workshop*, pages 18–25.

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.

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, pages 105–114, New York, NY, USA. ACM.

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*, pages 295–302. ACM.

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

Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. *arXiv preprint arXiv:1705.10513*.

Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-end neural ad-hoc ranking with kernel pooling. *arXiv preprint arXiv:1706.06613*.