End-to-End Neural Ad-hoc Ranking with Kernel Pooling

Chenyan Xiong*
Carnegie Mellon University
cx@cs.cmu.edu

Zhuyun Dai*
Carnegie Mellon University
zhuyund@cs.cmu.edu

Jamie Callan
Carnegie Mellon University
callan@cs.cmu.edu

Zhiyuan Liu
Tsinghua University
liuzy@tsinghua.edu.cn

Russell Power
Allen Institute for AI
russellp@allenai.org

ABSTRACT

This paper proposes K-NRM, a kernel based neural model for document ranking. Given a query and a set of documents, K-NRM uses a translation matrix that models word-level similarities via word embeddings, a new kernel-pooling technique that uses kernels to extract multi-level soft match features, and a learning-to-rank layer that combines those features into the final ranking score. The whole model is trained end-to-end. The ranking layer learns desired feature patterns from the pairwise ranking loss. The kernels transfer the feature patterns into soft-match targets at each similarity level and enforce them on the translation matrix. The word embeddings are tuned accordingly so that they can produce the desired soft matches. Experiments on a commercial search engine's query log demonstrate the improvements of K-NRM over prior feature-based and neural-based states-of-the-art, and explain the source of K-NRM's advantage: Its kernel-guided embedding encodes a similarity metric tailored for matching query words to document words, and provides effective multi-level soft matches.

KEYWORDS

Ranking, Neural IR, Kernel Pooling, Relevance Model, Embedding

1 INTRODUCTION

In traditional information retrieval, queries and documents are typically represented by discrete bags-of-words, the ranking is based on exact matches between query and document words, and trained ranking models rely heavily on feature engineering. In comparison, newer neural information retrieval (neural IR) methods use continuous text embeddings, model the query-document relevance via soft matches, and aim to learn feature representations automatically. With the successes of deep learning in many related areas, neural IR has the potential to redefine the boundaries of information retrieval; however, achieving that potential has been difficult so far.

Many neural approaches use distributed representations (e.g., word2vec [17]), but in spite of many efforts, distributed representations have a limited history of success for document ranking. Exact match of query words to document words is a strong signal of relevance [8], whereas soft-match is a weaker signal that must be used carefully. Word2vec may consider 'pittsburgh' to be similar to 'boston', and 'hotel' to be similar to 'motel'. However, a person searching for 'pittsburgh hotel' may accept a document about 'pittsburgh motel', but probably will reject a document about 'boston hotel'. How to use these soft-match signals effectively and reliably is an open problem.

This work addresses these challenges with a kernel based neural ranking model (K-NRM). K-NRM uses distributed representations to represent query and document words. Their similarities are used to construct a translation model. Word pair similarities are combined by a new kernel-pooling layer that uses kernels to softly count the frequencies of word pairs at different similarity levels (soft-TF). The soft-TF signals are used as features in a ranking layer, which produces the final ranking score. All of these layers are differentiable and allow K-NRM to be optimized end-to-end.

The kernels are the key to K-NRM's capability. During learning, the kernels convert the learning-to-rank loss to requirements on soft-TF patterns, and adjust the word embeddings to produce a soft match that can better separate the relevant and irrelevant documents. This kernel-guided embedding learning encodes a similarity metric tailored for matching query and document. The tailored similarity metric is conveyed by the learned embeddings, which produces effective multi-level soft-matches for ad-hoc ranking.

Extensive experiments on a commercial search engine's query log demonstrate the significant and robust advantage of K-NRM. On different evaluation scenarios (in-domain, cross-domain and raw user clicks), and on different parts of the query log (head, torso, and tail), K-NRM outperforms both feature-based ranking and neural ranking states-of-the-art by as much as 65%. K-NRM's advantage is not from an unexplainable 'deep learning magic', but the long-desired soft match achieved by its kernel-guided embedding learning. In our analysis, if used without the multi-level soft match or the embedding learning, the advantage of K-NRM quickly diminishes; while with the kernel-guided embedding learning, K-NRM successfully learns relevance-focused soft matches using its embedding and ranking layers, and the memorized ranking preferences generalize well to different testing scenarios.

The next section discusses related work. Section 3 presents the kernel-based neural ranking model. Experimental methodology is discussed in Section 4 and evaluation results are presented in Section 5. Section 6 concludes.

*The first two authors contributed equally.

SIGIR '17, August 07-11, 2017, Shinjuku, Tokyo, Japan

DOI: http://dx.doi.org/10.1145/3077136.3080809

2 RELATED WORK

Retrieval models such as query likelihood and BM25 are based on exact matching of query and document words, which limits the information available to the ranking model and may lead to problems such *vocabulary mismatch* [4]. *Statistical translation models* were an attempt to overcome this limitation. They model query-document relevance using a pre-computed *translation matrix* that describes the similarities between word pairs [1]. At query time, the ranking function considers the similarities of all query and document word pairs, allowing query words to be soft-matched to document words. The translation matrix can be calculated via mutual information in a corpus [12] or using user clicks [6].

Word pair interactions have also been modeled by *word embeddings*. Word embeddings trained from surrounding contexts, for example, word2vec [17], are considered to be the factorization of word pairs' PMI matrix [14]. Compared to word pair similarities which are hard to learn, word embeddings provide a smooth low-level approximation of word similarities that may improve translation models [8, 24].

Some research has questioned whether word embeddings based on surrounding context, such as word2vec, are suitable for ad hoc ranking. Instead, it customizes word embeddings for search tasks. Nalisnick et al. propose to match query and documents using both the input and output of the embedding model, instead of only using one side of them [19]. Diaz et al. find that word embeddings trained locally on pseudo relevance feedback documents are more related to the query's information needs, and can provide better query expansion terms [5].

Current neural ranking models fall into two groups: *representation* based and *interaction* based [8]. The earlier focus of neural IR was mainly on *representation* based models, in which the query and documents are first embedded into continuous vectors, and the ranking is calculated from their embeddings' similarity. For example, DSSM [11] and its convolutional version CDSSM [22] map words to letter-tri-grams, embed query and documents using neural networks built upon the letter-tri-grams, and rank documents using their embedding similarity with the query.

The *interaction* based neural models, on the other hand, learn query-document matching patterns from word-level interactions. For example, ARC-II [10] and MatchPyramid [20] build hierarchical Convolutional Neural Networks (CNN) on the interactions of two texts' word embeddings; they are effective in matching tweet-retweet and question-answers [10]. The Deep Relevance Matching Model (DRMM) uses pyramid pooling (histogram) [7] to summarize the word-level similarities into ranking signals [9]. The word level similarities are calculated from pre-trained word2vec embeddings, and the histogram counts the number of word pairs at different similarity levels. The counts are combined by a feed forward network to produce final ranking scores. *Interaction* based models and *representation* based models address the ranking task from different perspectives, and can be combined for further improvements [18].

This work builds upon the ideas of customizing word embeddings and the *interaction* based neural models: K-NRM ranks documents using soft matches from query-document word interactions, and learns to encode the relevance preferences using customized word embeddings at the same time, which is achieved by the kernels.

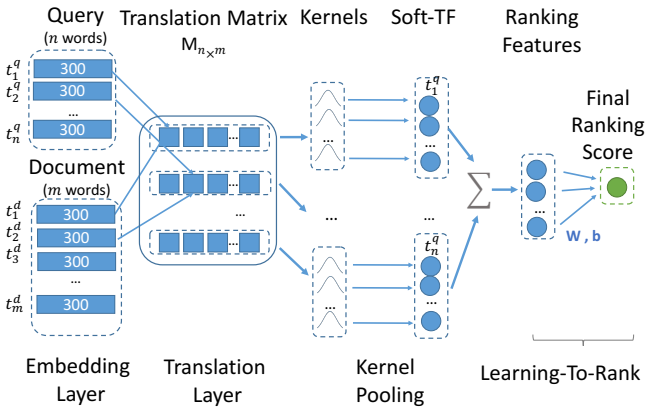


Figure 1: The Architecture of K-NRM. Given input query words and document words, the embedding layer maps them into distributed representations, the translation layer calculates the word-word similarities and forms the translation matrix, the kernel pooling layer generate soft-TF counts as ranking features, and the learning to rank layer combines the soft-TF to the final ranking score.

3 KERNEL BASED NEURAL RANKING

This section presents K-NRM, our kernel based neural ranking model. We first discuss how K-NRM produces the ranking score for a query-document pair with their words as the sole input (ranking from scratch). Then we derive how the ranking parameters and word embeddings in K-NRM are trained from ranking labels (learning end-to-end).

3.1 Ranking from Scratch

Given a query q and a document d, K-NRM aims to generate a ranking score $f(q, d)$ only using query words $q = \{t_1^q, ...t_i^q ..., t_n^q\}$ and document words $d = \{t_1^d, ...t_j^d ..., t_m^d\}$. As shown in Figure 1, K-NRM achieves this goal via three components: translation model, kernel-pooling, and learning to rank.

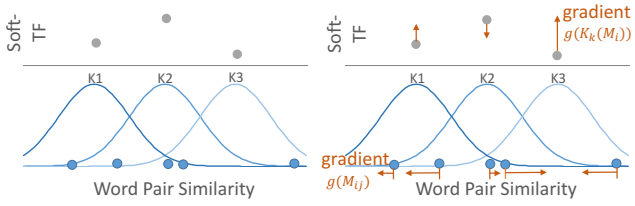
Translation Model: K-NRM first uses an `embedding layer` to map each word t to an L-dimension embedding $\vec{v}_t$:

$$t \Rightarrow \vec{v}_t .$$

Then a `translation layer` constructs a translation matrix M. Each element in M is the embedding similarity between a query word and a document word:

$$M_{ij} = \cos(\vec{v}_{t_i^q}, \vec{v}_{t_j^d}).$$

The translation model in K-NRM uses word embeddings to recover the word similarities instead of trying to learn one for each word pair. Doing so requires much fewer parameters to learn. For a vocabulary of size $|V|$ and the embedding dimension L, K-NRM's translation model includes $|V| \times L$ embedding parameters, much fewer than learning all pairwise similarities ($|V|^2$).



(a) Ranking **(b) Learning**

Figure 2: Illustration of kernels in the ranking (forward) process and learning (backward) process.

Kernel-Pooling: K-NRM then uses kernels to convert word-word interactions in the translation matrix M to query-document ranking features $\phi(M)$:

$$\phi(M) = \sum_{i=1}^{n} \log \vec{K}(M_i)$$
$$\vec{K}(M_i) = \{K_1(M_i), ..., K_K(M_i)\}$$

$\vec{K}(M_i)$ applies K kernels to the i-th query word's row of the translation matrix, summarizing (pooling) it into a K-dimensional feature vector. The log-sum of each query word's feature vector forms the query-document ranking feature vector ϕ.

The effect of $\vec{K}$ depends on the kernel used. This work uses the RBF kernel:

$$K_k(M_i) = \sum_j \exp(-\frac{(M_{ij} - \mu_k)^2}{2\sigma_k^2}).$$

As illustrated in Figure 2a, the RBF kernel K_k calculates how word pair similarities are distributed around it: the more word pairs with similarities closer to its mean μ_k, the higher its value. Kernel pooling with RBF kernels is a generalization of existing pooling techniques. As $\sigma \to \infty$, the kernel pooling function devolves to the mean pooling. $\mu = 1$ and $\sigma \to 0$ results in a kernel that only responds to exact matches, equivalent to the TF value from sparse models. Otherwise, the kernel functions as 'soft-TF'[1]. μ defines the similarity level that 'soft-TF' focuses on; for example, a kernel with $\mu = 0.5$ calculates the number of document words whose similarities to the query word are close to 0.5. σ defines the kernel width, or the range of its 'soft-TF' count.

Learning to Rank: The ranking features $\phi(M)$ are combined by a ranking layer to produce the final ranking score:

$$f(q, d) = \tanh(w^T \phi(M) + b).$$

w and b are the ranking parameters to learn. tanh() is the activation function. It controls the range of ranking score to facilitate the learning process. It is rank-equivalent to a typical linear learning to rank model.

[1]The RBF kernel is one of the most popular choices. Other kernels with similar density estimation effects can also be used, as long as they are differentiable. For example, polynomial kernel can be used, but histograms [9] cannot as they are not differentiable.

Putting every together, K-NRM is defined as:

$$f(q, d) = \tanh(w^T \phi(M) + b) \quad \text{Learning to Rank} \quad (1)$$
$$\phi(M) = \sum_{i=1}^{n} \log \vec{K}(M_i) \quad \text{Soft-TF Features} \quad (2)$$
$$\vec{K}(M_i) = \{K_1(M_i), ..., K_K(M_i)\} \quad \text{Kernel Pooling} \quad (3)$$
$$K_k(M_i) = \sum_j \exp(-\frac{(M_{ij} - \mu_k)^2}{2\sigma_k^2}) \quad \text{RBF Kernel} \quad (4)$$
$$M_{ij} = \cos(\vec{v}_{t_i^q}, \vec{v}_{t_j^d}) \quad \text{Translation Matrix} \quad (5)$$
$$t \Rightarrow \vec{v}_t. \quad \text{Word Embedding} \quad (6)$$

Eq. 5-6 embed query words and document words, and calculate the translation matrix. The kernels (Eq. 4) count the soft matches between query and document's word pairs at multiple levels, and generate K soft-TF ranking features (Eq. 2-3). Eq. 1 is the learning to rank model. The ranking of K-NRM requires no manual features. The only input used is the query and document words. The kernels extract soft-TF ranking features from word-word interactions automatically.

3.2 Learning End-to-End

The **training** of K-NRM uses the pairwise learning to rank loss:

$$l(w, b, \mathcal{V}) = \sum_q \sum_{d^+, d^- \in D_q^{+,-}} max(0, 1 - f(q, d^+) + f(q, d^-)). \quad (7)$$

$D_q^{+,-}$ are the pairwise preferences from the ground truth: d^+ ranks higher than d^-. The parameters to learn include the ranking parameters w, b, and the word embeddings $\mathcal{V}$.

The parameters are optimized using back propagation (BP) through the neural network. Starting from the ranking loss, the gradients are first propagated to the learning-to-rank part (Eq. 1) and update the ranking parameters (w, b), the kernels pass the gradients to the word similarities (Eq. 2-4), and then to the embeddings (Eq. 5).

Back propagations through the kernels: The embeddings contain millions of parameters $\mathcal{V}$ and are the main capacity of the model. The learning of the embeddings is guided by the kernels.

The back propagation first applies gradients from the loss function (Eq. 7) to the ranking score $f(q, d)$, to increase it (for d^+) or decrease it (for d^-); the gradients are propagated through Eq. 1 to the feature vector $\phi(M)$, and then through Eq. 2 to the the kernel scores $\vec{K}(M_i)$. The resulted $g(\vec{K}(M_i))$ is a K dimensional vector:

$$g(\vec{K}(M_i)) = \{g(K_1(M_i)), ..., g(K_K(M_i))\}.$$

Its each dimension $g(K_k(M_i))$ is jointly defined by the ranking score's gradients and the ranking parameters. It adjusts the corresponding kernel's score up or down to better separate the relevant document (d^+) from the irrelevant one (d^-).

The kernels spread the gradient to word similarities in the translation matrix M_{ij}, through Eq. 4:

$$g(M_{ij}) = \sum_{k=1}^{K} \frac{g(K_k(M_i)) \times \sigma_k^2}{(\mu_k - M_{ij}) \exp(\frac{(M_{ij} - \mu_k)^2}{-2\sigma_k^2})}. \quad (8)$$

The kernel-guided embedding learning process is illustrated in Figure 2b. A kernel pulls the word similarities closer to its μ to

Table 1: Training and testing dataset characteristics.

	Training	Testing
Queries	95,229	1,000
Documents Per Query	12.17	30.50
Search Sessions	31,201,876	4,103,230
Vocabulary Size	165,877	19,079

increase its soft-TF count, or pushes the word pairs away to reduce it, based on the gradients received in the back-propagation. The *strength* of the force also depends on the the kernel's width σ_k and the word pair's distance to μ_k: approximately, the wider the kernel is (bigger σ_k), and the closer the word pair's similarity to μ_k, the stronger the force is (Eq. 8). The gradient a word pair's similarity received, $g(M_{ij})$, is the combination of the forces from all K kernels.

The word embedding model receives $g(M_{ij})$ and updates the embeddings accordingly. Intuitively, the learned word embeddings are aligned to form multi-level soft-TF patterns that can separate the relevant documents from the irrelevant ones in training, and the learned embedding parameters $\mathcal{V}$ memorize this information. When testing, K-NRM extracts soft-TF features from the learned word embeddings using the kernels and produces the final ranking score using the ranking layer.

4 EXPERIMENTAL METHODOLOGY

This section describes our experimental methods and materials.

4.1 Dataset

Our experiments use a query log sampled from search logs of Sogou.com, a major Chinese commercial search engine. The sample contains 35 million search sessions with 96,229 distinct queries. The query log includes queries, displayed documents, user clicks, and dwell times. Each query has an average of 12 documents displayed. As the results come from a commercial search engine, the returned documents tend to be of very high quality.

The primary testing queries were 1,000 queries sampled from head queries that appeared more than 1,000 times in the query log. Most of our evaluation focuses on the head queries; we use tail query performance to evaluate model robustness. The remaining queries were used to train the neural models. Table 1 provides summary statistics for the training and testing portions of the search log.

The query log contains only document titles and URLs. The full texts of *testing* documents were crawled and parsed using Boilerpipe [13] for our word-based baselines (described in Section 4.3). Chinese text was segmented using the open source software ICTCLAS [23]. After segmentation, documents are treated as sequences of words (as with English documents).

4.2 Relevance Labels and Evaluation Scenarios

Neural models like K-NRM and CDSSM require a large amount of training data. Acquiring a sufficient number of manual training labels outside of a large organization would be cost-prohibitive. User click data, on the other hand, is easy to acquire and prior research has shown that it can accurately predict manual labels. For our experiments **training labels** were generated based on user clicks from the training sessions.

Table 2: Testing Scenarios. DCTR Scores are inferred by DCTR click model [3]. TACM Scores are inferred by TACM click model [15]. Raw Clicks use the sole click in a session as the positive label. The label distribution is the number of relevance labels from 0-4 from left to right, if applicable.

Condition	Label	Label Distribution
Testing-SAME	DCTR Scores	70%, 19.6%, 9.8%, 1.3%, 1.1%
Testing-DIFF	TACM Scores	79%, 14.7%, 4.6%, 0.9%, 0.9%
Testing-RAW	Raw Clicks	2,349,280 clicks

There is a large amount of prior research on building *click models* to model user behavior and to infer reliable relevance signals from clicks [3]. This work uses one of the simplest click models, DCTR, to generate relevance scores from user clicks [3]. DCTR calculates the relevance scores of a query-document pair based on their click through rates. Despite being extremely simple, it performs rather well and is a widely used baseline [3]. Relevance scores from DCTR are then used to generate preference pairs to train our models.

The **testing labels** were also estimated from the click log, as manual relevance judgments were not made available to us. *Note that there was no overlap between training queries and testing queries.*

Testing-SAME infers relevance labels using DCTR, the same click model used for training. This setting evaluates the ranking model's ability to fit user preferences (click through rates).

Testing-DIFF infers relevance scores using TACM [15], a state-of-the-art click model. TACM is a more sophisticated model and uses both clicks and dwell times. On an earlier sample of Sogou's query log, the TACM labels aligned extremely well with expert annotations: when evaluated against manual labels, TACM achieved an NDCG@5 of up to 0.97 [15]. This is substantially higher than the agreement between the manual labels generated by the authors for a sample of queries. This precision makes TACM's inferred scores a good approximation of expert labels, and Testing-DIFF is expected to produce evaluation results similar to expert labels.

Testing-RAW is the simplest click model. Following the cascade assumption [3], we treat the clicked document in each single-click session as a relevant document, and test whether the model can put it at the top of the ranking. Testing-Raw only uses single-click sessions (57% of the testing sessions are single-click sessions). Testing-RAW is a conservative setting that uses *raw* user feedback. It eliminates the influence of click models in testing, and evaluates the ranking model's ability to overcome possible disturbances from the click models.

The three testing scenarios are listed in Table 2. Following TREC methodology, the Testing-SAME and Testing-DIFF's inferred relevance scores were mapped to 5 relevance grades. Thresholds were chosen so that our relevance grades have the same distribution as TREC Web Track 2009-2012 qrels.

Search quality was measured using NDCG at depths $\{1, 3, 10\}$ for Testing-SAME and Testing-DIFF. We focused on early ranking positions that are more important for commercial search engines. Testing-RAW was evaluated by mean reciprocal rank (MRR) as there is only one relevant document per query. Statistical significance was tested using the permutation test with $p < 0.05$.

Table 3: The number of parameters and the word embeddings used by baselines and K-NRM. '–' indicates not applicable, e.g. unsupervised methods have no parameters, and word-based methods do not use embeddings.

Method	Number of Parameters	Embedding
Lm, BM25	–	–
RankSVM	21	–
Coor-Ascent	21	–
Trans	–	word2vec
DRMM	161	word2vec
CDSSM	10,877,657	–
K-NRM	49,763,110	end-to-end

4.3 Baselines

Our baselines include both traditional word-based ranking models as well as more recent neural ranking models.

Word-based baselines include BM25 and language models with Dirichlet smoothing (Lm). These unsupervised retrieval methods were applied on the full text of candidate documents, and used to re-rank them. We found that these methods performed better on full text than on titles. Full text default parameters were used.

Feature-based learning to rank baselines include RankSVM[2], a state-of-the-art pairwise ranker, and coordinate ascent [16] (Coor-Ascent[3]), a state-of-the-art listwise ranker. They use typical word-based features: Boolean AND; Boolean OR; Coordinate match; TF-IDF; BM25; language models with no smoothing, Dirichlet smoothing, JM smoothing and two-way smoothing; and bias. All features were applied to the document title and body. The parameters of the retrieval models used in feature extraction are kept default.

Neural ranking baselines include DRMM [9], CDSSM [21], and a simple embedding-based translation model, Trans.

DRMM is the state-of-the-art interaction based neural ranking model [9]. It performs histogram pooling on the embedding based translation matrix and uses the binned soft-TF as the input to a ranking neural network. The embeddings used are pre-trained via word2vec [17] because the histograms are not differentiable and prohibit end-to-end learning. We implemented the best variant, $\text{DRMM}_{\text{LCH×IDF}}$. The pre-trained embeddings were obtained by applying the skip-gram method from word2vec on our training corpus (document titles displayed in training sessions).

CDSSM [22] is the convolutional version of DSSM [11]. CDSSM maps English words to letter-tri-grams using a word-hashing technique, and uses Convolutional Neural Networks to build representations of the query and document upon the letter-tri-grams. It is a state-of-the-art representation based neural ranking model. We implemented CDSSM in Chinese by convolving over Chinese characters. (Chinese characters can be considered as similar to English letter-tri-grams with respect to word meaning). CDSSM is also an end-to-end model, but uses discrete letter-tri-grams/Chinese characters instead of word embeddings.

Trans is an unsupervised embedding based translation model. Its translation matrix is calculated by the cosine similarity of word embeddings from the same word2vec used in DRMM, and then averaged to the query-document ranking score.

Baseline Settings: RankSVM uses a linear kernel and the hyper-parameter C was selected in the development fold of the cross validation from the range $[0.0001, 10]$.

Recommended settings from RankLib were used for Coor-Ascent.

We obtained the body texts of testing documents from the new Sogou-T corpus [2] or crawled them directly. The body texts were used by all word-based baselines. Neural ranking baselines and K-NRM used only titles for training and testing, as the coverage of Sogou-T on the training documents is low and the training documents could not be crawled given resource constraints.

For all baselines, the most optimistic choices were made: feature-based methods (RankSVM and Coor-Ascent) were trained using 10-fold cross-validation on the *testing* set and use both document title and body texts. The neural models were trained on the training set with the same settings as K-NRM, and only use document titles (they still perform better than only using the testing data). Theoretically, this gives the sparse models a slight performance advantage as their training and testing data were drawn from the same distribution.

4.4 Implementation Details

This section describes the configurations of our K-NRM model.

Model training was done on the full training data as in Table 1, with training labels inferred by DCTR, as described in Section 4.2.

The **embedding layer** used 300 dimensions. The vocabulary size of our training data was $165,877$. The embedding layer was initialized with the word2vec trained on our training corpus.

The **kernel pooling layer** had $K = 11$ kernels. One kernel harvests exact matches, using $\mu_0 = 1.0$ and $\sigma = 10^{-3}$. μ of the other 10 kernels is spaced evenly in the range of $[-1, 1]$, that is $\mu_1 = 0.9, \mu_2 = 0.7, ..., \mu_{10} = -0.9$. These kernels can be viewed as 10 soft-TF bins. σ is set to 0.1. The effects of varying σ are studied in Section 5.6.

Model optimization used the Adam optimizer, with batch size 16, learning rate = 0.001 and $\epsilon = 1e-5$. Early stopping was used with a patience of 5 epochs. We implemented our model using TensorFlow. The model training took about 50 milliseconds per batch, and converged in 12 hours on an AWS GPU machine.

Table 3 summarizes the number of parameters used by the baselines and K-NRM. Word2vec refers to pre-trained word embeddings using skip-gram on the training corpus. End-to-end means that the embeddings were trained together with the ranking model.

CDSSM learns hundreds of convolution filters on Chinese *characters*, thus has millions of parameters. K-NRM's parameter space is even larger as it learns an embedding for every Chinese *word*. Models with more parameters in general are expected to fit better but may also require more training data to avoid overfitting.

5 EVALUATION RESULTS

Our experiments investigated K-NRM's effectiveness, as well as its behavior on tail queries, with less training data, and with different kernel widths.

[2] https://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html
[3] https://sourceforge.net/p/lemur/wiki/RankLib/

Table 4: Ranking accuracy of K-NRM and baseline methods. Relative performances compared with Coor-Ascent are in percentages. Win/Tie/Loss are the number of queries improved, unchanged, or hurt, compared to Coor-Ascent on NDCG@10. †, ‡, §, ¶ indicate statistically significant improvements over Coor-Ascent†, Trans‡, DRMM§ and CDSSM¶, respectively.

(a) Testing-SAME. Testing labels are inferred by the same click model (DCTR) as the training labels used by neural models.

Method	NDCG@1		NDCG@3		NDCG@10		W/T/L
Lm	0.1261	−20.89%	0.1648	−26.46%	0.2821	−20.45%	293/116/498
BM25	0.1422	−10.79%	0.1757	−21.60%	0.2868	−10.14%	299/125/483
RankSVM	0.1457	−8.59%	0.1905	−14.99%	0.3087	−12.97%	371/151/385
Coor-Ascent	0.1594‡§¶	–	0.2241‡§¶	–	0.3547‡§¶	–	–/–/–
Trans	0.1347	−15.50%	0.1852	−17.36%	0.3147	−11.28%	318/140/449
DRMM	0.1366	−14.30%	0.1902	−15.13%	0.3150	−11.20%	318/132/457
CDSSM	0.1441	−9.59%	0.2014	−10.13%	0.3329‡§	−6.14%	341/149/417
K-NRM	**0.2642**†‡§¶	+65.75%	**0.3210**†‡§¶	+43.25%	**0.4277**†‡§¶	+20.58%	447/153/307

(b) Testing-DIFF. Testing labels are inferred by a different click model, TACM, which approximates expert labels very well [15].

Method	NDCG@1		NDCG@3		NDCG@10		W/T/L
Lm	0.1852	−11.34%	0.1989	−17.23%	0.3270	−13.38%	369/50/513
BM25	0.1631	−21.92%	0.1894	−21.18%	0.3254	−13.81%	349/53/530
RankSVM	0.1700	−18.62%	0.2036	−15.27%	0.3519	−6.78%	380/75/477
Coor-Ascent	0.2089‡¶	–	0.2403‡	–	0.3775‡¶	–	–/–/–
Trans	0.1874	−10.29%	0.2127	−11.50%	0.3454	−8.51%	385/68/479
DRMM	0.2068	−1.00%	0.2491‡	+3.67%	0.3809‡¶	+0.91%	430/66/436
CDSSM	0.1846	−10.77%	0.2358‡	−1.86%	0.3557	−5.79%	391/65/476
K-NRM	**0.2984**†‡§¶	+42.84%	**0.3092**†‡§¶	+28.26%	**0.4201**†‡§¶	+11.28%	474/63/395

Table 5: Ranking performance on Testing-RAW. MRR evaluates the mean reciprocal rank of clicked documents in single-click sessions. Relative performance in the percentages and W(in)/T(ie)/L(oss) are compared to Coor-Ascent. †, ‡, §, ¶ indicate statistically significant improvements over Coor-Ascent†, Trans‡, DRMM§ and CDSSM¶, respectively.

Method	MRR		W/T/L
Lm	0.2193	−9.19%	416/09/511
BM25	0.2280	−5.57%	456/07/473
RankSVM	0.2241	−7.20%	450/78/473
Coor-Ascent	0.2415‡	–	–/–/–
Trans	0.2181	−9.67%	406/08/522
DRMM	0.2335‡	−3.29%	419/12/505
CDSSM	0.2321‡	−3.90%	405/11/520
K-NRM	**0.3379**†‡§¶	+39.92%	507/05/424

5.1 Ranking Accuracy

Tables 4a, 4b and 5 show the ranking accuracy of K-NRM and our baselines under three conditions.

Testing-SAME (Table 4a) evaluates the model's ability to fit user preferences when trained and evaluated on labels generated by the same click model (DCTR). K-NRM outperforms word-based baselines by over 65% on NDCG@1, and over 20% on NDCG@10. The improvements over neural ranking models are even bigger: On NDCG@1 the margin between K-NRM and the next best neural model is 83%, and on NDCG@10 it is 28%.

Testing-DIFF (Table 4b) evaluates the model's relevance matching performance by testing on TACM inferred relevance labels, a good approximation of expert labels. Because the training and testing labels were generated by different click models, Testing-DIFF challenges each model's ability to fit the underlying relevance signals despite perturbations caused by differing click model biases. Neural models with larger parameter spaces tend to be more vulnerable to this domain difference: CDSSM actually performs worse than DRMM, despite using thousands times more parameters. However, K-NRM demonstrates its robustness and is able to outperform all baselines by more than 40% on NDCG@1 and 10% on NDCG@10.

Testing-RAW (Table 5) evaluates each model's effectiveness directly by user clicks. It tests how well the model ranks the most satisfying document (the only one clicked) in each session. K-NRM improves MRR from 0.2415 (Coor-Ascent) to 0.3379. This difference is equal to moving the clicked document's from rank 4 to rank 3. The MRR and NDCG@1 improvements demonstrate K-NRM's precision oriented property—its biggest advantage is on the earliest ranking positions. This characteristic aligns with K-NRM's potential role in web search engines: as a sophisticate re-ranker, K-NRM is most possibly used at the final ranking stage, in which the first relevant document's ranking position is the most important.

The two neural ranking baselines DRMM and CDSSM perform similarly in all three testing scenarios. The *interaction* based model, DRMM, is more robust to click model biases and performs slightly better on Testing-DIFF, while the *representation* based model, CDSSM, performs slightly better on Testing-SAME. However, the feature-based ranking model, Coor-Ascent, performs better than all neural

Table 6: The ranking performances of several K-NRM variants. Relative performances and statistical significances are all compared with K-NRM's full model. †, ‡, §, ¶, and ∗ indicate statistically significant improvements over K-NRM's variants of exact-match†, word2vec‡, click2vec§, max-pool¶, and mean-pool∗, respectively.

	Testing-SAME				Testing-DIFF				Testing-RAW	
K-NRM Variant	**NDCG@1**		**NDCG@10**		**NDCG@1**		**NDCG@10**		**MRR**	
exact-match	0.1351	−49%	0.2943	−31%	0.1788	−40%	$0.3460^{¶}$	−18%	0.2147	−37%
word2vec	$0.1529^{†}$	−42%	$0.3223^{†¶}$	−24%	$0.2160^{†¶}$	−27%	$0.3811^{†¶}$	−10%	$0.2427^{†¶}$	−28%
click2vec	0.1600	−39%	$0.3790^{†‡¶}$	−11%	$0.2314^{†¶}$	−23%	$0.4002^{†‡¶*}$	−4%	$0.2667^{†‡¶}$	−21%
max-pool	0.1413	−47%	0.2979	−30%	0.1607	−46%	0.3334	−21%	0.2260	−33%
mean-pool	$0.2297^{†‡§¶}$	−13%	$0.3614^{†‡§¶}$	−16%	$0.2424^{†¶}$	−19%	$0.3787^{†¶}$	−10%	$0.2714^{†‡¶}$	−20%
full model	$\mathbf{0.2642}^{†‡§¶*}$	–	$\mathbf{0.4277}^{†‡§¶*}$	–	$\mathbf{0.2984}^{†‡§¶*}$	–	$\mathbf{0.4201}^{†‡¶*}$	–	$\mathbf{0.3379}^{†‡§¶*}$	–

baselines on all three testing scenarios. The differences can be as high as 15% and some are statistically significant. This holds even for Testing-SAME which is expected to favor deep models that access more in-domain training data. These results remind that no 'deep learning magic' can instantly provide significant gains for information retrieval tasks. The development of neural IR models also requires an understanding of the advantages of neural methods and how their advantages can be incorporated to meet the needs of information retrieval tasks.

Table 7: Examples of word matches in different kernels. Words in bold are those whose similarities with the query word fall into the corresponding kernel's range (μ).

μ	Query: 'Maserati' "
1.0	**Maserati** Ghibli black interior _ who knows
0.7	Maserati **Ghibli** black interior _ who knows
0.3	Maserati Ghibli **black** interior _ **who** knows
-0.1	Maserati Ghibli black interior _ who **knows**

5.2 Source of Effectiveness

K-NRM differs from previous ranking methods in several ways: multi-level soft matches, word embeddings learned directly from ranking labels, and the kernel-guided embedding learning. This experiment studies these effects by comparing the following variants of K-NRM.

K-NRM (exact-match) only uses the exact match kernel $(\mu, \sigma) = (1, 0.001)$. It is equivalent to TF.

K-NRM (word2vec) uses pre-trained word2vec, the same as DRMM. Word embedding is fixed; only the ranking part is learned.

K-NRM (click2vec) also uses pre-trained word embedding. But its word embeddings are trained on (query word, clicked title word) pairs. The embeddings are trained using skip-gram model with the same settings used to train word2vec. These embeddings are fixed during ranking.

K-NRM (max-pool) replaces kernel-pooling with max-pooling. Max-pooling finds the maximum similarities between document words and each query word; it is commonly used by neural network architectures. In our case, given the candidate documents' high quality, the maximum is almost always 1, thus it is similar to TF.

K-NRM (mean-pool) replaces kernel-pooling with mean-pooling. It is similar to Trans except that the embedding is trained by learning-to-rank.

All other settings are kept the same as K-NRM. Table 6 shows their evaluation results, together with the full model of K-NRM.

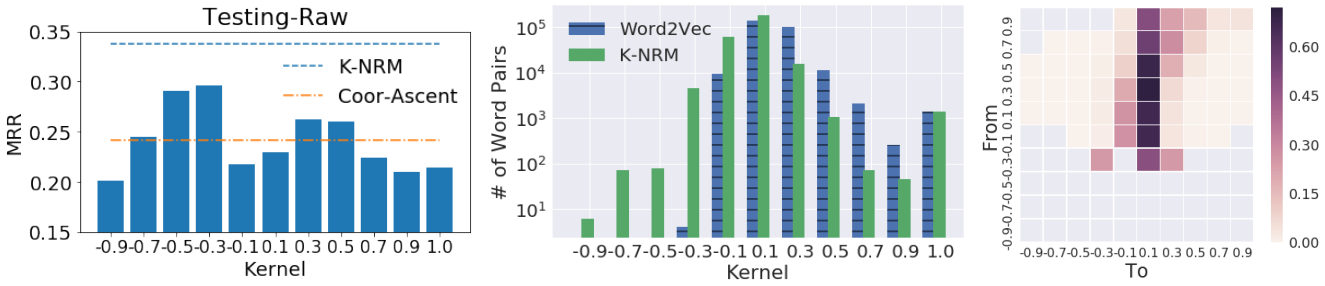
Soft match is essential. K-NRM (exact-match) performs similarly to Lm and BM25, as does K-NRM (max-pool). This is expected: without soft matches, the only signal for K-NRM to work with is effectively the TF score.

Ad-hoc ranking prefers relevance based word embedding. Using click2vec performs about 5-10% better than using word2vec. User clicks are expected to be a better fit as they represent user search preferences, instead of word usage in documents. The relevance-based word embedding is essential for neural models to outperform feature-based ranking. K-NRM (click2vec) consistently outperforms Coor-Ascent, but K-NRM (word2vec) does not.

Learning-to-rank trains better word embeddings. K-NRM with mean-pool performs much better than Trans. They both use average embedding similarities; the difference is that K-NRM (mean-pool) uses the ranking labels to tune the word embeddings, while Trans keeps the embeddings fixed. The trained embeddings improve the ranking accuracy, especially on top ranking positions.

Kernel-guided embedding learning provides better soft matches. K-NRM stably outperforms all of its variants. K-NRM (click2vec) uses the same ranking model, and its embeddings are trained on click contexts. K-NRM (mean-pool) also learns the word embeddings using learning-to-rank. The main difference is how the information from relevant labels is used when learning word embeddings. In KNRM (click2vec) and K-NRM (mean-pool), training signals from relevance labels are propagated *equally* to all query-document word pairs. In comparison, K-NRM uses kernels to enforce multi-level soft matches; query-document word pairs on different similarity levels are adjusted differently (see Section 3.2).

Table 7 shows an example of K-NRM's learned embeddings. The **bold** words in each row are those 'activated' by the corresponding kernel: their embedding similarities to the query word 'Maserati' fall closest to the kernel's μ. The example illustrates that the kernels recover different levels of relevance matching: $\mu = 1$ is exact match; $\mu = 0.7$ matches the car model with the brand; $\mu = 0.3$ is about the car color; $\mu = -0.1$ is background noise. The mean-pool and click2vec's uni-level training loss mix the matches at multiple levels, while the kernels provide more fine-grained training for the embeddings.



(a) Individual kernel's performance.

(b) Kernel sizes before and after learning.

(c) Word pair movements.

Figure 3: Kernel guided word embedding learning in K-NRM. Fig. 3a shows the performance of K-NRM when only one kernel is used in *testing*. Its X-axis is the μ of the used kernel. Its Y-axis is the MRR results. Fig. 3b shows the log number of word pairs that are closest to each kernel, before K-NRM learning (Word2Vec) and after. Its X-axis is the μ of kernels. Fig. 3c illustrates the word pairs' movements in K-NRM's learning. The heat map shows the fraction of word pairs from the row kernel (before learning, μ marked on the left) to the column kernel (after learning, μ at the bottom).

5.3 Kernel-Guided Word Embedding learning

In K-NRM, word embeddings are initialized by word2vec and trained by the kernels to provide effective soft-match patterns. This experiment studies how training affects the word embeddings, showing the responses of kernels in ranking, the word similarity distributions, and the word pair movements during learning.

Figure 3a shows the performance of K-NRM when only a single kernel is used during *testing*. The x-axis is the μ of the kernel. The results indicate the kernels' importance. The kernels on the far left (≤ -0.7), the far right (≥ 0.7), and in the middle ($\{-0.1, 0.1\}$) contribute little; the kernels on the middle left ($\{-0.3, -0.5\}$) contribute the most, followed by those on the middle right ($\{0.3, 0.5\}$). Higher μ does not necessarily mean higher importance or better soft matching. Each kernel focuses on a group of word pairs that fall into a certain similarity range; the importance of this similarity range is learned by the model.

Figure 3b shows the number of word pairs activated in each kernel before training (Word2Vec) and after (K-NRM). The X-axis is the kernel's μ. The Y-axis is the log number of word pairs activated (whose similarities are closest to corresponding kernel's μ). Most similarities fall into the range (-0.4, 0.6). These histograms help explain why the kernels on the far right and far left do not contribute much: because there are fewer word pairs in them.

Figure 3c shows the word movements during the embedding learning. Each cell in the matrix contains the word pairs whose similarities are moved from the kernel in the corresponding row (μ on the left) to the kernel in the corresponding column (μ at the bottom). The color indicates the fraction of the moved word pairs in the original kernel. Darker indicates a higher fraction. Several examples of word movements are listed in Table 8. Combining Figure 3 and Table 8, the following trends can be observed in the kernel-guided embedding learning process.

Many word pairs are decoupled. Most of the word movements are from other kernels to the 'white noise' kernels $\mu \in \{-0.1, 0.1\}$. These word pairs are considered related by word2vec but not by K-NRM. This is the most frequent effect in K-NRM's embedding learning. Only about 10% of word pairs with similarities ≥ 0.5 are kept. This implies that document ranking requires a stricter measure of soft match. For example, as shown in Table 8's first row, a person searching for 'China-Unicom', one of the major mobile carriers in China, is less likely interested in a document about 'China-Mobile', another carrier; in the second row, 'Maserati' and 'car' are decoupled as 'car' appears in almost all candidate documents' titles, so it does not provide much evidence for ranking.

New soft match patterns are discovered. K-NRM moved some word pairs from near zero similarities to important kernels. As shown in the third and fourth rows of Table 8, there are word pairs that less frequently appear in the same surrounding context, but convey possible search tasks, for example, 'the *search* for *MH370* '. K-NRM also discovers word pairs that convey strong 'irrelevant' signals, for example, people searching for 'BMW' are not interested in the 'contact us' page.

Different levels of soft matches are enforced. Some word pairs moved from one important kernel to another. This may reflect the different levels of soft matches K-NRM learned. Some examples are in the last two rows in Table 8. The -0.3 kernel is the most important one, and received word pairs that encode search tasks; the 0.5 kernel received synonyms, which are useful but not the most important, as exact match is not that important in our setting.

5.4 Required Training Data Size

This experiment studies K-NRM's performance with varying amounts of training data. Results are shown in Figure 4. The X-axis is the number of sessions used for training (e.g. $8K, 32K, \ldots$), and the coverage of testing vocabulary in the learned embedding (percentages). Sessions were randomly sampled from the training set. The Y-axis is the performance of the corresponding model. The straight and dotted lines are the performances of Coor-Ascent.

When only 2K training sessions are available, K-NRM performs worse than Coor-Ascent. Its word embeddings are mostly unchanged from word2vec as only 16% of the testing vocabulary are covered by the training sessions. K-NRM's accuracy grows rapidly with more training sessions. With only 32K (0.1%) training sessions and 50% coverage of the testing vocabulary, K-NRM surpasses Coor-Ascent on Testing-RAW. With 128K (0.4%) training sessions and 69% coverage on the testing vocabularies, K-NRM surpasses

Table 8: Examples of moved word pairs in K-NRM. From and To are the μ of the kernels the word pairs were in before learning (word2vec) and after (K-NRM). Values in parenthesis are the individual kernel's MRR on Testing-RAW, indicating the importance of the kernel. '+' and '–' mark the sign of the kernel weight w_k in the ranking layer; '+' means word pair appearances in the corresponding kernel are positively correlated with relevance; '–' means negatively correlated.

From	To	Word Pairs
$\mu = 0.9$ (0.20, –)	$\mu = 0.1$ (0.23, –)	(wife, husband), (son, daughter), (China-Unicom, China-Mobile)
$\mu = 0.5$ (0.26, –)	$\mu = 0.1$ (0.23, –)	(Maserati, car),(first, time) (website, homepage)
$\mu = 0.1$ (0.23, –)	$\mu = -0.3$ (0.30, +)	(MH370, search), (pdf, reader) (192.168.0.1, router)
$\mu = 0.1$ (0.23, –)	$\mu = 0.3$ (0.26, –)	(BMW, contact-us), (Win7, Ghost-XP)
$\mu = 0.5$ (0.26, –)	$\mu = -0.3$ (0.30, +)	(MH370, truth), (cloud, share) (HongKong, horse-racing)
$\mu = -0.3$ (0.30, +)	$\mu = 0.5$ (0.26, –)	(oppor9, OPPOR), (6080, 6080YY), (10086, www.10086.com)

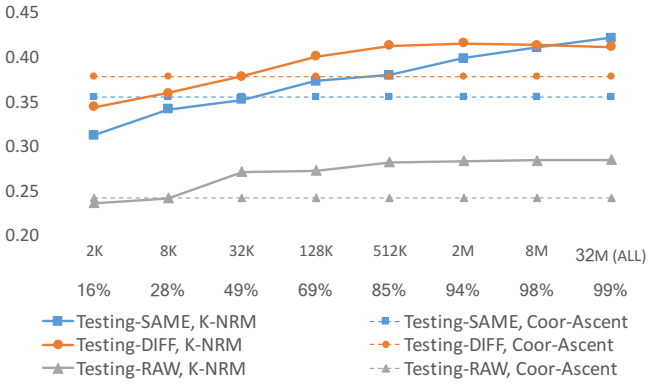


Figure 4: K-NRM's performances with different amounts of training data. X-axis: Number of sessions used for training, and the percentages of testing vocabulary covered (second row). Y-axis: NDCG@10 for Testing-SAME and Testing-DIFF, and MRR for Testing-RAW.

Coor-Ascent on Testing-SAME and Testing-DIFF. The increasing trends against Testing-SAME and Testing-RAW have not yet plateaued even with 31M training sessions, suggesting that K-NRM can utilize more training data. The performance on Testing-DIFF plateaus after 500K sessions, perhaps because the click models do not perfectly align with each other; more regularization of the K-NRM model might help under this condition.

5.5 Performance on Tail Queries

This experiment studies how K-NRM performs on less frequent queries. We split the queries in the query log into Tail (less than 50 appearances), Torso (50-1000 appearances), and Head (more than 1000 appearances). For each category, 1000 queries are randomly sampled as testing; the remaining queries are used for training. Following the same experimental settings, the ranking accuracies of K-NRM and Coor-Ascent are evaluated.

Table 9: Ranking accuracy on Tail (frequency < 50), Torso (frequency $50 - 1K$) and Head (frequency $> 1K$) queries. † indicates statistically significant improvements of K-NRM over Coor-Ascent on Testing-RAW. Frac is the fraction of the corresponding queries in the search traffic. Cov is the fraction of testing query words covered by the training data.

	Frac	Cov	Testing-RAW, MRR		
			Coor-Ascent	K-NRM	
Tail	52%	85%	0.2977	0.3230†	+8.49%
Torso	20%	91%	0.3202	0.3768†	+17.68%
Head	28%	99%	0.2415	0.3379†	+39.92%

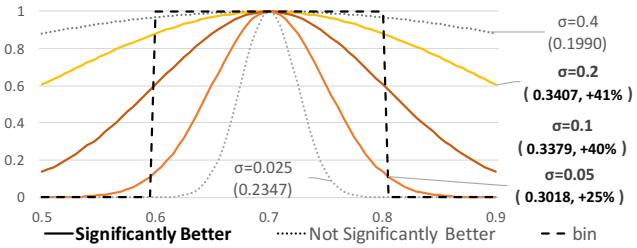


Figure 5: K-NRM's performance with different σ. MRR and relative gains over Coor-Ascent are shown in parenthesis. Kernels drawn in solid lines indicate statistically significant improvements over Coor-Ascent.

The results are shown in Table 9. Evaluation is only done using Testing-RAW as the tail queries do not provide enough clicks for DCTR and TACM to infer reliable relevance scores. The results show an expected decline of K-NRM's performance on rarer queries. K-NRM uses word embeddings to encode the relevance signals, and as tail queries' words appear less frequently in the training data, it is hard to generalize the embedded relevance signals through them. Nevertheless, even on queries that appear less than 50 times, K-NRM still outperforms Coor-Ascent by 8%.

5.6 Hyper Parameter Study

This experiment studies the influence of the kernel width (σ). We varied the σ used in K-NRM's kernels, kept everything else unchanged, and evaluated its performance. The shapes of the kernels with 5 different σ and the corresponding ranking accuracies are shown in Figure 5. Only Testing-RAW is shown due to limited space; the observation is the same on Testing-SAME and Testing-DIFF.

As shown in Figure 5, kernels too sharp or too flat either do not cover the similarity space well, or mixed the matches at different levels; they cannot provide reliable improvements. With σ between 0.05 and 0.2, K-NRM's improvements are stable.

We have also experimented with several other structures for K-NRM, for example, using more learning to rank layers, and using IDF to weight query words when combining their kernel-pooling

results. However we have only observed similar or worse performances. Thus, we chose to present the simplest successful model to better illustrate the source of its effectiveness.

6 CONCLUSION

This paper presents K-NRM, a kernel based neural ranking model for ad-hoc search. The model captures word-level interactions using word embeddings, and ranks documents using a learning-to-rank layer. The center of the model is the new kernel-pooling technique. It uses kernels to softly count word matches at different similarity levels and provide soft-TF ranking features. The kernels are differentiable and support end-to-end learning. Supervised by ranking labels, the learning of word embeddings is guided by the kernels with the goal of providing soft-match signals that better separate relevant documents from irrelevant ones. The learned embeddings encode the relevance preferences and provide effective multi-level soft matches for ad-hoc ranking.

Our experiments on a commercial search engine's query log demonstrated K-NRM's advantages. On three testing scenarios (in-domain, cross-domain, and raw user clicks), K-NRM outperforms both feature based ranking baselines and neural ranking baselines by as much as 65%, and is extremely effective at the top ranking positions. The improvements are also robust: Stable gains are observed on head and tail queries, with fewer training data, a wide range of kernel widths, and a simple ranking layer.

Our analysis revealed that K-NRM's advantage is not from 'deep learning magic' but the long-desired soft match between query and documents, which is achieved by the kernel-guided embedding learning. Without it, K-NRM's advantage quickly diminishes: its variants with only exact match, pre-trained word2vec, or uni-level embedding training all perform significantly worse, and sometimes fail to outperform the simple feature based baselines.

Further analysis of the learned embeddings illustrated how K-NRM tailors them for ad-hoc ranking: More than 90% of word pairs that are mapped together in word2vec are decoupled, satisfying the stricter definition of soft match required in ad-hoc search. Word pairs that are less correlated in documents but convey frequent search tasks are discovered and mapped to certain similarity levels. The kernels also moved word pairs from one kernel to another based on their different roles in the learned soft match.

Our experiments and analysis not only demonstrated the effectiveness of K-NRM, but also provide useful intuitions about the advantages of neural methods and how they can be tailored for IR tasks. We hope our findings will be explored in many other IR tasks and will inspire more advances of neural IR research in the near future.

7 ACKNOWLEDGMENTS

This research was supported by National Science Foundation (NSF) grant IIS-1422676, a Google Faculty Research Award, and a fellowship from the Allen Institute for Artificial Intelligence. We thank Tie-Yan Liu for his insightful comments and Cheng Luo for helping us crawl the testing documents. Any opinions, findings, and conclusions in this paper are the authors' and do not necessarily reflect those of the sponsors.

REFERENCES

[1] A. Berger and J. Lafferty. Information retrieval as statistical translation. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 222–229. ACM, 1999.

[2] L. Cheng, Z. Yukun, L. Yiqun, X. Jingfang, Z. Min, and M. Shaoping. SogouT-16: A new web corpus to embrace ir research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, page To Appear. ACM, 2017.

[3] A. Chuklin, I. Markov, and M. d. Rijke. Click models for web search. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 7(3):1–115, 2015.

[4] W. B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Reading, 2010.

[5] F. Diaz, B. Mitra, and N. Craswell. Query expansion with locally-trained word embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational (ACL)*. ACL–Association for Computational Linguistics, 2016.

[6] J. Gao, X. He, and J.-Y. Nie. Clickthrough-based translation models for web search: from word models to phrase models. In *Proceedings of the 19th ACM international conference on Information and knowledge management (CIKM)*, pages 1139–1148. ACM, 2010.

[7] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *Tenth IEEE International Conference on Computer Vision (ICCV) Volume 1*, volume 2, pages 1458–1465. IEEE, 2005.

[8] J. Guo, Y. Fan, Q. Ai, and W. B. Croft. Semantic matching by non-linear word transportation for information retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM)*, pages 701–710. ACM, 2016.

[9] J. Guo, Y. Fan, A. Qingyao, and W. B. Croft. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM)*, pages 55–64, year=2016, organization=ACM.

[10] B. Hu, Z. Lu, H. Li, and Q. Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2042–2050, 2014.

[11] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management (CIKM)*, pages 2333–2338. ACM, 2013.

[12] M. Karimzadehgan and C. Zhai. Estimation of statistical translation models based on mutual information for ad hoc information retrieval. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 323–330. ACM, 2010.

[13] C. Kohlschütter, P. Fankhauser, and W. Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the third ACM international conference on Web Search and Data Mining (WSDM)*, pages 441–450. ACM, 2010.

[14] O. Levy, Y. Goldberg, and I. Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

[15] Y. Liu, X. Xie, C. Wang, J.-Y. Nie, M. Zhang, and S. Ma. Time-aware click model. *ACM Transactions on Information Systems (TOIS)*, 35(3):16, 2016.

[16] D. Metzler and W. B. Croft. Linear feature-based models for information retrieval. *Information Retrieval*, 10(3):257–274, 2007.

[17] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 2tth Advances in Neural Information Processing Systems 2013 (NIPS)*, pages 3111–3119, 2013.

[18] B. Mitra, F. Diaz, and N. Craswell. Learning to match using local and distributed representations of text for web search. In *Proceedings of the 25th International Conference on World Wide Web (WWW)*, pages 1291–1299. ACM, 2017.

[19] E. Nalisnick, B. Mitra, N. Craswell, and R. Caruana. Improving document ranking with dual word embeddings. In *Proceedings of the 25th International Conference on World Wide Web (WWW)*, pages 83–84. ACM, 2016.

[20] L. Pang, Y. Lan, J. Guo, J. Xu, S. Wan, and X. Cheng. Text matching as image recognition. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI, pages 2793–2799. AAAI Press, 2016.

[21] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management (CIKM)*, pages 101–110. ACM, 2014.

[22] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web (WWW)*, pages 373–374. ACM, 2014.

[23] H.-P. Zhang, H.-K. Yu, D.-Y. Xiong, and Q. Liu. HHMM-based chinese lexical analyzer ICTCLAS. In *Proceedings of the second SIGHAN workshop on Chinese language processing*, pages 184–187. ACL, 2003.

[24] G. Zuccon, B. Koopman, P. Bruza, and L. Azzopardi. Integrating and evaluating neural word embeddings in information retrieval. In *Proceedings of the 20th Australasian Document Computing Symposium*, page 12. ACM, 2015.