Efficient Interaction-based Neural Ranking with Locality Sensitive Hashing

Shiyu Ji
Department of Computer Science,
University of California
Santa Barbara, California

Jinjin Shao
Department of Computer Science,
University of California
Santa Barbara, California

Tao Yang
Department of Computer Science,
University of California
Santa Barbara, California

ABSTRACT

Interaction-based neural ranking has been shown to be effective for document search using distributed word representations. However the time or space required is very expensive for online query processing with neural ranking. This paper investigates fast approximation of three interaction-based neural ranking algorithms using Locality Sensitive Hashing (LSH). It accelerates query-document interaction computation by using a runtime cache with precomputed term vectors, and speeds up kernel calculation by taking advantages of limited integer similarity values. This paper presents the design choices with cost analysis, and an evaluation that assesses efficiency benefits and relevance tradeoffs for the tested datasets.

ACM Reference Format:
Shiyu Ji, Jinjin Shao, and Tao Yang. 2019. Efficient Interaction-based Neural Ranking with Locality Sensitive Hashing. In *Proceedings of the 2019 World Wide Web Conference (WWW'19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3308558.4090049

1 INTRODUCTION AND RELATED WORK

Neural networks have been applied in information retrieval and text mining tasks (e.g. [5, 9, 20, 23, 24, 33–36]). In general, exploiting neural representation of words adds significant rank scoring computation cost during query processing. While parallel computing with a large number of CPUs or GPUs can accelerate, this paper studies how to speed up neural rank scoring with an algorithmic approximation to reach a reasonable query response time without resorting to expensive hardware. This is important for a large online query processing website which needs to dedicate more hardware resource for handling high query traffic. Fast scoring also opens opportunities to integrate a more complex neural scheme.

The previous research on neural ranking falls into the following two categories: interaction-based or representation-based models [23]. The earlier work has focused on the representation based models [11, 27] where each document and a query are separately represented as vectors through a sequence of neural computations and the final ranking is based on the similarity of the two representative vectors. The recent studies have focused on interaction-based neural ranking models [5, 9, 34] where the word or term-level similarity of a query and a document is explored first based on their embedding vectors before applying an additional sequence of neural computation. These studies have shown their interaction based models outperform the earlier representation-based models and thus our paper is focused on the acceleration of the three interaction-based models, more specifically DRMM [9], KNRM [34], and CONV-KNRM [5]. We find that the query processing time cost in the above interaction-based models for computing query-document interaction and deriving kernel values is dominating. For CONV-KNRM, the time to prepare term vectors is very expensive.

The contribution of this paper is an LSH approximation of these three neural methods with fast histogram-based kernel calculation and term vector precomputing for a runtime cache. Another contribution is an analysis and experimental evaluation of design choices and a demonstration using more complex dual embeddings leveraging fast neural scoring. While our study is focused on the above three models, LSH approximation may be used for other interaction-based methods that employ cosine similarity (e.g. graph embedding [35, 36]) and this is a future work.

Our work is motivated by the rich history of research on approximation with hashing such as LSH [1–4, 7, 8, 13, 15, 37] and in various data-intensive applications [6, 10, 17, 18, 30]. The reason to choose the Random Hyperplane LSH [4] is that the computation with hashed embeddings can not only preserve the semantic similarity within a small error bound, but also allow us to take advantage of the binary nature of sketch vectors for designing a fast kernel calculation.

The acceleration of neural network computation has been studied for image classification with complex neural network structure (e.g. [12, 26, 28] for binarization). Their work is less applicable to our context where neural network involved is relatively simple and cost of forward neural computation in query processing is one or two orders of magnitudes smaller than interaction and kernel computation. To our knowledge, there is no research work on accelerating neural ranking models for fast online query processing.

2 INTERACTION-BASED NEURAL RANKING

Interaction-based neural ranking can be formalized as performing computation: RankingScore = $f(Ker(\vec{q} \otimes \vec{d}))$, where $\vec{q}$ and $\vec{d}$ are two sequences of embedding vectors for the query and the document respectively, $\otimes$ is the interaction operation, Ker is the operation for kernel computation, and f is the operation involving neural network computation. Embedding vectors in $\vec{q}$ and $\vec{d}$ can be term vectors for unigram/n-gram in query/document [5, 9, 34], or can be entity vectors for existing entities in query/document [35]. Those embedding vectors can be learned from one of many existing neural embedding models [19, 25] or generated by additional neural operations [5].

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.4090049

Table 1: Frequently used notations.

Notation	Definition
$\vec{q}$	Embedding vectors for query terms.
$\vec{d}$	Embedding vectors for document terms.
m	Number of words in the query.
n	Number of words in the document.
l	Dimension of pre-trained unigram word embedding vectors.
h	The largest span of n-gram considered in CONV-KNRM [5].
F	The number of filters in CONV-KNRM [5].
M	The interaction matrix between query and document.
V	Vocabulary size of unigrams in the corpus.
H	Vocabulary size of n-grams in the corpus where $n \leq h$.
N	Number of documents in the corpus.
R	Number of kernels in the neural ranking model.
μ_k	Mean of the k-th kernel in KNRM and CONV-KNRM.
σ_k	Standard deviation of the k-th kernel.
$K_k(M_i)$	The k-th kernel value for the i-th query term.
b	Number of bits in hashing footprint.

Interaction operator $\otimes$. This step outputs an interaction matrix M containing vector similarities for all pairs of one query term vector and one document term vector.

- **DRMM [9] and KNRM [34]** consider only the interaction between unigrams in the query and unigrams in the document. Each matrix element M_{ij} is the cosine similarity between the term vectors of the i-th query term and the j-th document term.
- **CONV-KNRM [5]** extends KNRM to incorporate n-grams in building a set of interaction matrices. Let h be the maximum n-gram length considered and there are h^2 interaction matrices. Each interaction matrix $M^{r,t}$ where $1 \leq r, t \leq h$ denotes a cosine similarity score matrix between query term vector modeled with r-gram embeddings and document term vectors modeled with t-gram embeddings. These n-gram embeddings are computed from convolution filters and pre-trained unigram embedding vectors. The dimension of n-gram embeddings depends on the number of filters involved (defined as F).

Since computing each interaction matrix requires a dot product computation for each matrix element, given a document with n terms, the time complexity to compute the similarities of this document for m-word query is $\Theta(nml)$ for DRMM and KNRM, and $\Theta(nmF)$ for CONV-KNRM.

Kernel computation operator Ker. Ker is used for extracting multi-level soft matching features and generating inputs for neural network computation.

- In DRMM, each interaction matrix element represents the similarity between a query term and a document term, varying from -1 to 1. The interval $[-1, 1]$ is divided into a set of non-overlapping ranges. Thus a histogram can be generated where each bucket represents one range, and the value of a bucket represents the number of similarities falling into that bucket. Different buckets contain soft matching signals of different levels, just like kernels in the Kernel Pooling [34]. Thus in this paper, for simplicity, we denote each bucket as a kernel. Calculating all kernel/bucket values requires $\Theta(nm)$ time complexity.
- In KNRM and CONV-KNRM, each kernel is defined using a distribution and computation involved is called kernel pooling. There are R kernels defined in advance and symbols μ_k and σ_k denote the mean and standard deviation of the k-th kernel respectively. Each element value of an interaction matrix contributes kernel computation. For KNRM with one interaction matrix M, the kernel value of the k-th kernel for the i-th query term is:

$$K_k(M_i) = \sum_{j=1}^{n} \exp(-\frac{(M_{ij} - \mu_k)^2}{2\sigma_k^2})$$

  where n is the number of terms in the document. The output of kernel computation is a vector of R size: $(\sum_{i=1}^{m} \log K_1(M_i), \cdots, \sum_{i=1}^{m} \log K_R(M_i))^T$. Building this kernel for each query term requires $\Theta(nmR)$ time given there are R kernels.
  In CONV-KNRM, there are h^2 interaction matrices and the total cost of this kernel computation is $\Theta(nmRh^2)$. This expression does not include the cost of preparing term vectors and we discuss this issue further in Section 3.3.

Forward neural network computation f. This step uses a forward neural network to compute the final ranking score based on a vector of kernel values computed in the previous step.

- In DRMM, a multilayer perception network with one hidden layer of a constant size is used and the logarithms of the kernel/bucket values of the similarity ranges for the i-th query terms form an input vector. The final ranking score is a weighted linear combination of the neural network outputs of all query terms. The time cost of computing is in $\Theta(mR)$.
- In KNRM, a simple single-layer network is used and the input vector is a vector of size R computed in the previous step. The time cost for neural network computation is in $\Theta(R)$.
- In CONV-KNRM that extends KNRM, the formula of forward neural computation in this step is about the same as KNRM except that the output of the previous step produces a vector of dimension Rh^2 given h^2 interaction matrices and this vector is the input of this step. Thus the time cost for forward neural network computation is in $\Theta(Rh^2)$.

3 NEURAL RANKING WITH LSH

We summarize the time cost of above interaction-based neural ranking steps for query processing in Table 2. This section proposes three techniques to bring down time cost of query processing: approximation of term vectors with LSH in Section 3.1, fast kernel computing in Section 3.2, and precomputing of term vectors for CONV-KNRM in Section 3.3. The query-processing time cost after using our three techniques is listed in Table 2 also.

3.1 LSH for fast interaction computation

We adopt a hyperplane LSH [4], motivated by the previous work in clustering and nearest neighbor search [17, 30] that adopts such a scheme also. In [17, 30], multiple LSH signatures are produced for each document and the probability of similar documents falls into the one of LSH buckets is computable. In our context, we only use one LSH signature to approximate one term vector.

Table 2: Time complexity in Θ notion for ranking one single document with and without LSH.

Models	DRMM		KNRM		CONV-KNRM without precomputing		CONV-KNRM with precomputing	
With LSH?	No	Yes	No	Yes	No	Yes	No	Yes
Vectors	-	-	-	-	$(n+m)h^2lF$	$(n+m)(hbF+h^2lF)$	h^2mlF	$(mhbF+h^2mlF)$
Interaction	nml	nm	nml	nm	h^2nmF	h^2nm	h^2nmF	h^2nm
Kernel	nm	nm	nmR	$nm+bmR$	h^2nmR	$h^2(nm+bmR)$	h^2nmR	$h^2(nm+bmR)$
Neural	mR	mR	R	R	Rh^2	Rh^2	Rh^2	Rh^2

Table 3: Dominating factor of storage space cost in bytes with and without LSH using a reference or value based method.

Models	DRMM		KNRM		CONV-KNRM without precomputing		CONV-KNRM with precomputing	
With LSH?	No	Yes	No	Yes	No	Yes	No	Yes
Reference	$4nN+4Vl$	$4nN+Vb/8$	$4nN+4Vl$	$4nN+Vb/8$	$4nN+4Vl$	$4nN+4Vl$	$4hnN+4HF$	$4hnN+Hb/8+4Vl$
Value	$4nNl$	$nNb/8$	$4nNl$	$nNb/8$	$4nNl+4Vl$	$4nNl+4Vl$	$4hnFN+4HF$	$hnbN/8+4Vl$

Formally, for each embedding vector, we generate b-bit footprint

$$\text{LSH}(v)[i] = \begin{cases} 1 & r_i \cdot v > 0, \\ 0 & r_i \cdot v \le 0. \end{cases}$$

Here $\text{LSH}(v)[i]$ denotes the i-th bit of the LSH footprint of the vector v, and each vector r_i is independently sampled by a multivariate normal distribution from the vector space of the embeddings. The time complexity to compute the above LSH for an embedding is $\Theta(bl)$, where l is the embedding length. In particular, for CONV-KNRM we need to compute LSH for the n-gram vectors in the query, which needs $\Theta(mhbF)$, since query of m words has mh n-grams (s.t. $n \le h$), and the length of each n-gram vector is F.

Then we estimate the angle between two embeddings by the Hamming distance between their LSH footprints. Formally, given any two embeddings x and y with angle θ between them, estimator $\hat{\theta}$ is defined as $\hat{\theta} = \frac{\pi}{b} D(\text{LSH}(x), \text{LSH}(y))$, where $D(\cdot,\cdot)$ denotes Hamming distance (Eq. (2) in [30]). Modern hardware is fast at finding Hamming distance between two words, or equivalently, counting the set bits of the XORed words. The time complexity of deriving the Hamming distance that approximates the cosine similarity is $\Theta(nm)$ for one n-word document and m-word query.

The above approximation error is estimated as follows. Given any two embedding vectors u, v with angle θ between them, the probability that both hyperplane LSH footprints have the identical i-th bit is $\Pr[\text{LSH}(u)[i] = \text{LSH}(v)[i]] = 1 - \theta/\pi$ (also in [4, 17, 30]). Since each bit in the footprint is sampled independently, the Hamming distance between the LSH footprints is a random variable of Binomial distribution with $p = \theta/\pi$. Let b be the number of bits in the LSH footprint. Then the mean and variance of the Hamming distance are $\mathbb{E}[D(\text{LSH}(u), \text{LSH}(v))] = b\frac{\theta}{\pi}$, $\text{Var}[D(\text{LSH}(u), \text{LSH}(v))] = b\frac{\theta}{\pi}\left(1 - \frac{\theta}{\pi}\right)$. For the above angle estimator between u and v, $\mathbb{E}[\hat{\theta}] = \frac{\pi}{b}\mathbb{E}[D(\text{LSH}(u), \text{LSH}(v))] = \theta$. Note that for an unbiased estimator, mean squared error (MSE) equals to its variance [31]. Then:

$$\begin{aligned} MSE = \text{Var}[\hat{\theta}] &= \frac{\pi^2}{b^2}\text{Var}[D(\text{LSH}(u), \text{LSH}(v))] \\ &= \pi\frac{\theta}{b}\left(1 - \frac{\theta}{\pi}\right) = \frac{\pi^2}{b}\frac{\theta}{\pi}\left(1 - \frac{\theta}{\pi}\right) \le \frac{\pi^2}{4b}. \end{aligned}$$

For a relatively large b value, the upper bound of MSE is fairly small. For example, when $b = 256$, $MSE \le 0.00963$.

There is a line of work to derive feature hashing [32] that can preserve cosine similarity with the same error variance. But each hashed value in [32] is a sequence of float point numbers and thus using such a method only accomplishes the dimensionality reduction, does not significantly decrease the time complexity.

3.2 Fast kernel computation

Once interaction matrices are derived, kernel computation still takes significant time compared to other steps in neural ranking. Our key idea for the second optimization technique is to take advantages that the distinct number of similarity values is limited after the use of Hamming distances as an approximation. With this in mind, we transform the kernel computation for a histogram-based formula with a significantly lower computational cost. With b-bit hyperplane LSH-based footprints, since there are at most $b+1$ values of Hamming distance, we first build a histogram on Hamming distances, and then compute kernels based on the histogram values.

For interaction matrix M in KNRM with R kernels, each kernel can be computed as

$$\begin{aligned} K_k(M_i) &= \sum_{j=1}^{n} \exp\left(-\frac{(\cos\hat{\theta}_{ij} - \mu_k)^2}{2\sigma_k^2}\right) \\ &= \sum_{j=1}^{n} \exp\left(-\frac{(\cos(\frac{\pi}{b}D_{ij}) - \mu_k)^2}{2\sigma_k^2}\right) \\ &= \sum_{d=0}^{b} \underbrace{|\{j : D_{ij} = d\}|}_{\textbf{online computed}} \cdot \underbrace{\exp\left(-\frac{(\cos(\frac{\pi}{b}d) - \mu_k)^2}{2\sigma_k^2}\right)}_{\textbf{offline computed}}, \end{aligned}$$

where D_{ij} denotes the Hamming distance between the i-th query term footprint $\text{LSH}(\mathbf{q}[i])$ and the j-th document term footprint $\text{LSH}(\mathbf{d}[j])$. Set $\{j : D_{ij} = d\}$ is the histogram on Hamming distances between query and document terms. The above formula marks the part of computation conducted in the offline time.

The above histogram-based method reduces time complexity of kernel computing from $\Theta(nmR)$ for the original algorithm to

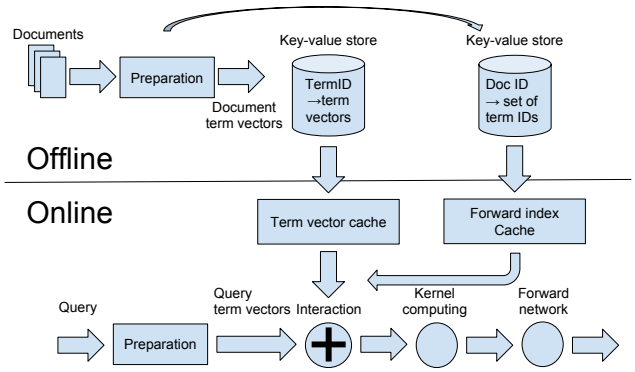

Figure 1: Architecture of interaction-based neural ranking.

$\Theta(nm+bmR)$. For web page ranking, a web page has 500 to 1K words on average, and neural ranking models usually take 20 to 50 kernels, there is a big cost difference. With $b = 256$, $n = 807$, and $R = 30$, the reduction ratio from above cost order expression $\Theta(nmR)$ to $\Theta(nm + bmR)$ is about 2.85. In our evaluation, we have observed around 417.38x speed improvement with the above histogram-based method. The reason for this large extra acceleration is partially because the original computation uses floating point arithmetic and our histogram-based method uses integer arithmetic. Another reason is that it allows us to re-arrange computation for better cache performance using a matrix notation and we will present more details on computation re-arrangement in an extended paper.

For CONV-KNRM with h^2 interaction matrices $M^{r,t}$ where $1 \leq r, t \leq h$, the above transformation is applied to each of R kernels for each of such matrices. Thus

$$K_k(M_i^{r,t}) = \sum_{d=0}^{b} \underbrace{|\{j : D_{ij}^{r,t} = d\}|}_{\textbf{online computed}} \cdot \underbrace{\exp(-\frac{(\cos(\frac{\pi}{b}d) - \mu_k)^2}{2\sigma_k^2})}_{\textbf{offline computed}}$$

where $D_{i,j}^{r,t}$ is the Hamming distance between the i-th LSH footprint of r-gram query term and the j-th LSH footprint of t-gram document term. The cost of kernel computation drops from the original complexity of $\Theta(h^2nmR)$ to $\Theta(h^2(nm + bmR))$.

3.3 Preparation of Term Vectors and Space Cost

For DRMM and KNRM, term vectors used for neural ranking directly adopt pretrained word embeddings (e.g. [19]), and thus there is no extra computation needed. In CONV-KNRM, a set of F convolution filters is applied to n-gram vectors based on word embeddings. The time complexity to prepare these document term vectors is $\Theta(nh^2lF)$ while deriving query term vectors costs $\Theta(mh^2lF)$.

Since $n >> m$, the time of this convolution computation for deriving document term vectors is dominating if it is conducted at the runtime when a query is handled. Our evaluation shows that without precomputing, CONV-KNRM spends too much time in preparing document term vectors. To reduce this cost, precomputation of these vectors can be employed to conduct such computation at the offline time. On the other hand, the disadvantage of precomputing is that it requires a substantial amount of space to store precomputed vectors, and LSH mapping can reduce this need significantly. The impact of incorporating LSH is described as follows.

- Without precomputing, following the original step of CONV-KNRM, we need to perform the convolution computation to derive all term vectors. Then we need extra $\Theta(mhbF)$ time to apply LSH to all unigram and n-gram term vectors for a query and documents to be ranked.
- With precomputing, the convolution computation and LSH mapping of all document term vectors are conducted in the offline time. The online query processing only needs to spend time to prepare query term vectors, which costs $\Theta(mhbF + h^2mlF)$.

Caching and space cost. Table 3 lists the dominating factors of disk space cost in bytes for the three algorithms with and without LSH or precomputing, assuming each term ID takes 32 bits and each floating number takes 32 bits. We describe two ways of storing document term representations as follows.

1) **Reference based method**.

Each term including a unigram or n-gram in the vocabulary is assigned an ID. Each term ID gives a reference to where the word embedding or term vector is stored. For each document, we need to store all the IDs of its necessary words or terms based on vector precomputing. Figure 1 illustrates the flow of neural ranking with two caches for runtime query processing. One cache is called the forward index cache which gives a mapping from a document ID to a set of term IDs this document owns. The second cache is called the term vector cache which is a mapping from a term ID to a term vector or its hash footprint. There is much higher access traffic to the second cache in ranking a document and thus the high hit ratio of the second cache is extremely important otherwise the cache miss would lead to a large number of random I/O operations to the disk.

2) **Value based method**. We store the values of embedding vectors or LSH footprints together with each document ID and they are directly accessible by a document ID. This method is useful when the memory cache demand is too high in the above reference method while it demands a larger amount of disk space.

For DRMM and KNRM with LSH, the main storage space cost for the reference method is about $4hnN$ to store the term IDs of documents and $Vb/8 + 4V$ bytes to store the map from term IDs to hash footprints where cost factor $4V$ is less significant. The second portion is the memory cache demand for fast access of hash footprints. The overall storage with LSH has a decent decrease compared to that without LSH. The size of the term-vector cache drops from about $4Vl$ without LSH to about $Vb/8$ with LSH. When $b = 256, l = 300$, the reduction ratio is about 32.8x.

For CONV-KNRM with LSH, if the reference based method is used with precomputing, the disk space needs $4hnN$ bytes for storing term IDs of N documents, and $Hb/8 + 4H$ bytes for hash footprints and index where $4H$ is less significant. Under precomputing, the size of the term-vector cache demand drops from about $4HF$ without LSH to about $Hb/8$ with LSH. When $F = 128, b = 256$, the cache space demand reduction ratio is about 6x.

To compare the storage cost of reference-based and value-based methods, using ClueWeb Category B dataset parameters with about 50 million documents, the dominating cost factor of storage space in CONV-KNRM with precomputing is $4hnN$ using the reference-based method and $hnbN/8$ using the value based method. The ratio of the value-based storage space cost over reference-based cost is approximately $\frac{b}{32}$, which is 8 when $b = 256$. With this size of the

database, the ratio of the value-based space cost over reference-based cost is also approximately $\frac{nNb/8}{4nN} = 8$ for DRMM with LSH and KNRM with LSH, respectively. With $b = 256$, the term vector disk space of the value-based method for CONV-KNRM is about 2.6TB for the ClueWeb dataset while the reference-based method costs about 351GB. Our evaluation uses the reference-based method.

4 EVALUATION

The objectives of our evaluation are to compare ranking relevance after applying LSH approximation with our design choices, demonstrate its time and space benefits, and assess the impact of using more complex embeddings on relevance and time cost.

Datasets and settings. We mainly report the performance using ClueWeb09 Category B with 50M webpages and 200 queries from the TREC Web Tracks 2009-2012 with 5-fold cross validation. We have also used two other smaller TREC datasets: TREC45 based on TREC Disks 4 and 5 with 0.5M news articles and AQUAINT with 1M news articles. Following the settings in [5, 9, 34], we first exclude spam webpages and rerank for top 1,000 documents of each query given by Galago initial ranking in the Lemur system [16]. For DRMM, the neural network has $R = 30$ kernels as input, and has a hidden layer of 5 neurons. For KNRM and CONV-KNRM, we do not update the embeddings during the training for simplicity, hence the accuracy scores may have a degradation compared with the results from the original papers. Other parameters used are: $l = 300$, $F = 128$, $h = 2$. We do not compare with the representation-based methods since the previous work shows that the interaction-based models perform better [23].

Word representation with dual embeddings. The previous work in DRMM, KNRM, CONV-KNRM typically used pretrained embeddings (e.g. word2vec) as a starting point. Nalisnick et al. firstly introduced Dual Embedding [21] for document ranking, given the fact that word2vec [19] with Negative Sampling actually trains two sets of word embedding vectors, defined as "input" (IN) and "output" (OUT) vector representations. Vector interactions between IN and IN are typically used in DRMM, KNRM, CONV-KNRM, and others for measuring term similarity. To leverage dual embedding interaction-based neural ranking models, we add a "dual" interaction matrix using IN vector representations and OUT vector representations and applying the same kernel computing techniques with the doubled time cost.

Ranking relevance and query processing time with hyperplane LSH. Table 4 shows the NDCG@20 scores [14] and average query time for different neural ranking models when using regular word embedding vectors or dual embeddings. The query processing C++ code for DRMM, KNRM, and CONV-KNRM is compiled with an optimization flag in a Linux machine which has Intel Xeon E5-2680v3 2.5 GHz dual socket and 128 GB DDR4 DRAM, and a local SSD with about 0.1 ms latency [29]. All the results are within confidence interval ±0.01 with p-value < 0.05. When increasing the number of bits for LSH, the NDCG score is getting closer to the original ones starting 64 bits, and the scores of 256 bits are fairly close to the original ones and we select this number as our choice in conducting other comparison. The competitiveness using LSH for NDCG@5 and @10 is similar. From Row 2, even time cost is roughly doubled, dual embeddings does yield a good improvement of relevance score compared to regular embeddings and LSH approximation can make it one or two orders of magnitude faster while retaining most of the relevance benefits from dual embeddings.

For average query time in Table 4, we use precomputing to derive the bigram and unigram embeddings for CONV-KNRM with and without LSH. The query response time in this table does not include the I/O cost to access document term embedding vectors. With 256 bits, DRMM with LSH is 4.12x faster than DRMM; KNRM with LSH 80.54x faster than KNRM; CONV-KNRM with LSH is 106.52x faster than CONV-KNRM. A similar speedup is accomplished when dual embeddings are used. Here we assume that term IDs of each document in processing a query are cached in memory. This forward index cache shown in Figure 1 does not need to be large and in the worst case with a cache miss, the I/O cost of accessing 1,000 such vectors is about 100ms from a local SSD. Thus we do not discuss the size demand of such a cache in this paper.

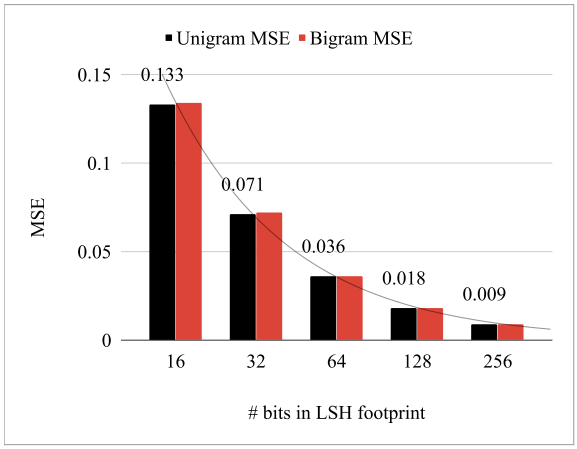

Figure 2: Errors of cosine similarity estimated with LSH.

To explain why the above three algorithms with LSH approximation can still be competitive in relevance when using a reasonable number of hash bits, Figure 2 shows that the mean squared errors of LSH-based cosine similarity approximation is relatively small for word2vec embeddings of unigrams and bigrams when increasing the number of hyperplane LSH bits. For example, when the number of LSH bits becomes 256, the actual approximation error is within 0.009, which corroborates that the relevance loss in Table 4 is reasonably small with such approximation.

Query time breakdown and impact of design optimization. With $b = 256$, Table 5 shows query time breakdown for different models in four steps discussed in Section 2 and Section 3 to re-rank 1,000 documents for a query for ClueWeb. Again document I/O cost is not included, which can be as high as 100ms per query. The underlined numbers are estimated since they require either too much time or too much space in experiments. For all methods, the time for forward neural computation step is relatively less significant compared with other steps. For DRMM, the interaction matrix computing time dominates and LSH produces a great reduction. For CONV-KNRM, the time cost for preparing term vectors corresponding word and n-gram embeddings is very expensive, and precomputing yields a significant advantage. For both KNRM and

Table 4: NDCG@20 scores and online query processing time in milliseconds.

Model	DRMM		KNRM		CONV-KNRM		Dual DRMM		Dual KNRM		Dual CONV-KNRM	
	NDCG@20	Time	NDCG@20	Time	NDCG@20	Time	NDCG@20	Time	NDCG@20	Time	NDCG@20	Time
Original	0.2580	140	0.2581	1,047	0.2688	4,900	0.2600	287	0.2665	2,073	0.2786	11,813
16-bit LSH	0.2449	7	0.2437	6	0.2563	13	0.2372	20	0.2371	7	0.2364	40
32-bit LSH	0.2464	13	0.2418	7	0.2521	18	0.2469	20	0.2557	13	0.2569	43
64-bit LSH	0.2489	20	0.2464	8	0.2571	20	0.2526	27	0.2563	26	0.2585	46
128-bit LSH	0.2528	27	0.2491	11	0.2619	27	0.2542	33	0.2633	27	0.2629	68
256-bit LSH	0.2518	34	0.2522	13	0.2640	46	0.2586	54	0.2627	33	0.2724	127
512-bit LSH	0.2551	48	0.2576	27	0.2680	86	0.2582	73	0.2624	80	0.2709	260
1024-bit LSH	0.2522	60	0.2561	53	0.2662	160	0.2573	140	0.2594	147	0.2753	548

Table 5: Query time breakdown in milliseconds for ranking 1000 documents.

Models	DRMM	KNRM	CONV-KNRM	CONV-KNRM Precomputing	DRMM LSH	KNRM LSH	CONV-KNRM LSH	CONV-KNRM Precomp. LSH
Vect. preparation	0	0	76,800.5	0.5	0	0	109,568.7	0.7
Interaction $\otimes$	121.0	50.6	1842.2	1842.2	13.7	9.7	32.0	32.0
Kernel Comp. Ker	10.1	993.2	3046.9	3046.9	11.8	1.8	7.3	7.3
Forward NN f	9.0	3.3	9.9	9.9	8.8	1.7	6.7	6.7
Total (ms)	140.1	1047.1	81,700.0	4899.5	34.3	13.2	109,614.7	46.7

Table 6: The size of disk storage and the memory cache for term vectors in GB.

Models	DRMM		KNRM		CONV-KNRM w/o precomputing		CONV-KNRM w/ precomputing	
With LSH?	No	Yes	No	Yes	No	Yes	No	Yes
TREC45	1.8 (0.8)	1.0 (0.02)	1.8 (0.8)	1.0 (0.02)	1.8 (0.8)	1.8 (0.8)	10.3 (8.3)	3.4 (1.3)
AQUAINT	2.7 (0.9)	1.8 (0.02)	2.7 (0.9)	1.8 (0.02)	2.7 (0.9)	2.7 (0.9)	12.5 (8.9)	5.1 (1.4)
ClueWeb09	178.4 (16.4)	162.5 (0.5)	178.4 (16.4)	162.5 (0.5)	178.4 (16.4)	178.4 (16.4)	492.4 (169.5)	351.0 (28.2)

CONV-KNRM, LSH reduces interaction matrix and kernel value computation significantly. The time for kernel computation is no longer dominating and our histogram-based kernel algorithm yields 417.38x reduction.

Table 6 compares the space requirements for three datasets with $b = 256$. The parameters of the datasets are listed blow. For the TREC45 data set, $N = 0.528$ million, average $n = 484$, $V = 0.67$ million, $H = 16.2$ million. For the AQUAINT data set, $N = 1.03$ million, $n = 440$, $V = 0.72$ million, $H = 17.3$ million. For the ClueWeb09 dataset, $N = 50.22$ million, $n = 807$, $V = 13.6$ million, $H = 328.43$ million [22]. The table lists two numbers in each entry. One is the total amount of disk storage space needed and another number inside a parenthesis is the size of the term-vector cache.

Our method using LSH reduces the size of term-vector cache significantly for DRMM/KNRM and CONV-KNRM with precomputing. For ClueWeb, the reference based method for CONV-KNRM without LSH needs about 492.4 GB in disk while memory size for term-vector cache is 169.5 GB. If such a memory cache is not available, the foot prints of each document need to be fetched with a random disk I/O. For ranking top 1000 documents matched for a query, the total I/O access cost can take about 100 seconds in our experiments. By integrating LSH, 351 GB disk space is still needed and the size of the term-vector cache is reduced by 6.01x to only 28.2 GB, which is fairly affordable.

5 CONCLUSION

This paper presents the design choices and algorithm optimization to improve the efficiency of three interaction-based neural ranking algorithms with LSH and a histogram-based kernel computing method. The evaluation shows that the proposed design choice yields 4.12x, 80.54x, and 106.52x time speedups for DRMM, KNRM, and CONV-KNRM respectively in the tested ClueWeb dataset with a reasonable response time and competitive relevance. Precomputing is used in our CONV-KNRM implementation for avoiding expensive document term vector preparation while it does carry extra space cost. Our LSH integration moderately decreases the overall storage space while making a major reduction for the term vector cache space demand, and the reduction ratio is 32.8x for DRMM and KNRM, and 6.01x for CONV-KNRM. This is critical to enable fast in-memory access of (hashed) embedding contents for a large dataset. More complex dual embeddings can improve relevance while its fast LSH approximation still preserves the most of its benefits.

ACKNOWLEDGMENTS

This work is supported in part by NSF IIS-1528041. It has used the NSF-supported resource in the Extreme Science and Engineering Discovery Environment (XSEDE) under allocation IRI190005 and

in the Center for Scientific Computing at UCSB, and we thank Michael Agun, Sharon Solis, and Mahidhar Tatineni for their help. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

[1] Alexandr Andoni and Piotr Indyk. 2006. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*. IEEE, 459–468.

[2] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. 2015. Practical and optimal LSH for angular distance. In *Advances in Neural Information Processing Systems*. 1225–1233.

[3] Alexandr Andoni and Ilya Razenshteyn. 2015. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. ACM, 793–801.

[4] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. ACM, 380–388.

[5] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. 2018. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 126–134.

[6] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 271–280.

[7] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 253–262.

[8] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*. 518–529.

[9] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. 2016. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of CIKM'16*. ACM, 55–64.

[10] Monika Henzinger. 2006. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 284–291.

[11] 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 Conference on information & knowledge management*. ACM, 2333–2338.

[12] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in neural information processing systems*. 4107–4115.

[13] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 604–613.

[14] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.* 20, 4 (Oct. 2002), 422–446.

[15] Brian Kulis and Trevor Darrell. 2009. Learning to hash with binary reconstructive embeddings. In *Advances in neural information processing systems*. 1042–1050.

[16] Lemur. [n. d.]. http://www.lemurproject.org/.

[17] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2014. *Mining of massive datasets*. Cambridge university press.

[18] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. 2007. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 141–150.

[19] 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*. 3111–3119.

[20] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1291–1299.

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

[22] Gregory B Newby, Chris Fallen, and Kylie McCormick. 2009. *Lucene for n-grams using the ClueWeb Collection*. Technical Report. ALASKA UNIV ANCHORAGE ARTIC REGION SUPERCOMPUTING CENTER.

[23] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, and Xueqi Cheng. 2017. A deep investigation of deep IR models. In *SIGIR 2017 Workshop on Neural Information Retrieval (Neu-IR'17)*.

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

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

[26] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer, 525–542.

[27] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 373–374.

[28] Wei Tang, Gang Hua, and Liang Wang. 2017. How to train a compact binary neural network with high accuracy?. In *AAAI*. 2625–2631.

[29] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr. 2014. XSEDE: Accelerating Scientific Discovery. *Computing in Science & Engineering* 16, 5 (Sept.-Oct. 2014), 62–74. https://doi.org/10.1109/MCSE.2014.80

[30] Ferhan Ture, Tamer Elsayed, and Jimmy Lin. 2011. No free lunch: brute force vs. locality-sensitive hashing for cross-lingual pairwise similarity. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. ACM, 943–952.

[31] Dennis Wackerly, William Mendenhall, and Richard L Scheaffer. 2014. *Mathematical statistics with applications*. Cengage Learning.

[32] Kilian Weinberger, Anirban Dasgupta, Josh Attenberg, John Langford, and Alex Smola. 2009. Feature hashing for large scale multitask learning. *arXiv preprint arXiv:0902.2206* (2009).

[33] Chenyan Xiong, Jamie Callan, and Tie-Yan Liu. 2017. Word-entity duet representations for document ranking. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 763–772.

[34] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 55–64.

[35] Chenyan Xiong, Zhengzhong Liu, Jamie Callan, and Tie-Yan Liu. 2018. Towards Better Text Understanding and Retrieval Through Kernel Entity Salience Modeling. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '18)*. ACM, 575–584.

[36] Chenyan Xiong, Russell Power, and Jamie Callan. 2017. Explicit semantic ranking for academic search via knowledge graph embedding. In *Proceedings of the 26th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 1271–1279.

[37] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. 2016. Deep Hashing Network for Efficient Similarity Retrieval.. In *AAAI*. 2415–2421.