A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval

Yelong Shen
Microsoft Research
Redmond, WA, USA
yeshen@microsoft.com

Xiaodong He
Microsoft Research
Redmond, WA, USA
xiaohe@microsoft.com

Jianfeng Gao
Microsoft Research
Redmond, WA, USA
jfgao@microsoft.com

Li Deng
Microsoft Research
Redmond, WA, USA
deng@microsoft.com

Grégoire Mesnil
University of Montréal
Montréal, Canada
gregoire.mesnil@umontreal.ca

ABSTRACT

In this paper, we propose a new latent semantic model that incorporates a convolutional-pooling structure over word sequences to learn low-dimensional, semantic vector representations for search queries and Web documents. In order to capture the rich contextual structures in a query or a document, we start with each word within a temporal context window in a word sequence to directly capture contextual features at the word n-gram level. Next, the salient word n-gram features in the word sequence are discovered by the model and are then aggregated to form a sentence-level feature vector. Finally, a non-linear transformation is applied to extract high-level semantic information to generate a continuous vector representation for the full text string. The proposed convolutional latent semantic model (CLSM) is trained on clickthrough data and is evaluated on a Web document ranking task using a large-scale, real-world data set. Results show that the proposed model effectively captures salient semantic information in queries and documents for the task while significantly outperforming previous state-of-the-art semantic models.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; I.2.6 [**Artificial Intelligence**]: *Learning*

General Terms

Algorithms, Experimentation

Keywords

Convolutional Neural Network, Semantic Representation, Web Search

1. INTRODUCTION

Most modern search engines resort to semantic based methods beyond lexical matching for Web document retrieval. This is partially due to the fact that the same single concept is often expressed using different vocabularies and language styles in documents and queries. For example, latent semantic models such as latent semantic analysis (LSA) are able to map a query to its relevant documents at the semantic level where lexical matching often fails (e.g., [9][10][31]). These models address the problem of language discrepancy between Web documents and search queries by grouping different terms that occur in a similar context into the same semantic cluster. Thus, a query and a document, represented as two vectors in the low-dimensional semantic space, can still have a high similarity even if they do not share any term. Extending from LSA, probabilistic topic models such as probabilistic LSA (PLSA), Latent Dirichlet Allocation (LDA), and Bi-Lingual Topic Model (BLTM), have been proposed and successfully applied to semantic matching [19][4][16][15][39]. More recently, semantic modeling methods based on neural networks have also been proposed for information retrieval (IR) [16][32][20]. Salakhutdinov and Hinton proposed the Semantic Hashing method based on a deep auto-encoder in [32][16]. A Deep Structured Semantic Model (DSSM) for Web search was proposed in [20], which is reported to give very strong IR performance on a large-scale web search task when clickthrough data are exploited as weakly-supervised information in training the model. In both methods, plain feed-forward neural networks are used to extract the semantic structures embedded in a query or a document.

Despite the progress made recently, all the aforementioned latent semantic models view a query (or a document) as a bag of words. As a result, they are not effective in modeling contextual structures of a query (or a document). Table 1 gives several examples of document titles to illustrate the problem. For example, the word "*office*" in the first document refers to the popular Microsoft product, but in the second document it refers to a working space. We see that the precise search intent of the word "*office*" cannot be identified without context.

microsoft *office* excel could allow remote code execution
welcome to the apartment *office*
online *body* fat percentage calculator
online auto *body* repair estimates

Table 1: Sample document titles. The text is lower-cased and punctuation removed. The same word, e.g., "*office*", has different meanings depending on its contexts.

Modeling contextual information in search queries and documents is a long-standing research topic in IR [11][25][12][26][2][22][24]. Classical retrieval models, such as TF-IDF and BM25, use a bag-of-words representation and cannot effectively capture contextual information of a word. Topic models learn the topic distribution of a word by considering word occurrence information within a document or a sentence. However, the contextual information captured by such models is

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CIKM'14, November 03 – 07, 2014, Shanghai, China.

Copyright © 2014 ACM 978-1-4503-2598-1/14/11…$15.00

http://dx.doi.org/10.1145/2661829.2661935

often too coarse-grained to be effective for the retrieval task. For example, the word *office* in "office excel" and "apartment office", which represent two very different search intents when used in search queries, are likely to be projected to the same topic. As an alternative, retrieval methods that directly model phrases (or word n-grams) and term dependencies are proposed in [12][25][26]. For example, in [25], the Markov Random Field (MRF) is used to model dependencies among terms (e.g., term n-grams and skip-grams) of the query and the document for ranking, while in [26] a latent concept expansion (LCE) model is proposed which leverages the term-dependent information by adding n-gram and (unordered n-gram) as features into the log-linear ranking model. In [12] a phrase-based translation model was proposed to learn the translation probability of a multi-term phrase in a query given a phrase in a document. Since the phrases capture richer contextual information than words, more precise translations can be determined. However, the phrase translation model can only score phrase-to-phrase pairs observed in the clickthrough training data and thus generalize poorly to new phrases.

In this study, we develop a new latent semantic model based on the convolutional neural network with convolution-pooling structure, called the *convolutional latent semantic model* (CLSM), to capture the important contextual information for latent semantic modeling. Instead of using the input representation based on bag-of-words, the new model views a query or a document[1] as a sequence of words with rich contextual structure, and it retains maximal contextual information in its projected latent semantic representation. The CLSM first projects each word within its context to a low-dimensional continuous feature vector, which directly captures the contextual features at the word n-gram level (detailed in section 3.3). Second, instead of summing over all word n-gram features uniformly, the CLSM discovers and aggregates only the salient semantic concepts to form a sentence-level feature vector (detailed in section 3.4). Then, the sentence-level feature vector is further fed to a regular feed-forward neural network, which performs a non-linear transformation, to extract high-level semantic information of the word sequence. In training, the parameters of the CLSM is learned on clickthrough data.

Our research contributions can be summarized as follows:

- We propose a novel CLSM that captures both the word n-gram level and sentence-level contextual structures for IR using carefully designed convolution and pooling operations;
- We carry out an extensive experimental study on the proposed model whereby several state-of-the-art semantic models are compared, and we achieve a significant performance improvement on a large-scale real-world Web search data set;
- We perform an in-depth case analysis on the capacity of the proposed model, through which the strength of the CLSM is clearly demonstrated.

[1] In modern search engines, a Web document is described by multiple fields [12][38], including title, body, anchor text etc. In our experiments, we only used the title field of a Web document for ranking. In addition to providing simplicity for fast experimentation, our decision is motivated by the observation that the title field gives better single-field retrieval result than body, although it is much shorter (as shown in Table 4). Thus it can serve as a reasonable baseline in our experiments. Nevertheless, our methods are not limited to the title field, and can be easily applied to the multi-field description.

2. RELATED WORK

2.1 Modeling Term Dependencies for IR

Although most traditional retrieval models assume the occurrences of terms to be completely independent, contextual information is crucial for detecting particular search intent of a query term. Thus, research in this area has been focusing on capturing term dependencies. Early work tries to relax the independence assumption by including phrases, in addition to single terms, as indexing units [6][36]. Phrases are defined by collocations (adjacency or proximity) and selected on the statistical ground, possibly with some syntactic knowledge. Unfortunately, the experiments did not provide a clear indication whether the retrieval effectiveness can be improved in this way. Recently, within the framework of language models for IR, various approaches that go beyond unigrams have been proposed to capture certain term dependencies, notably the bigram and tri-gram models [35], the dependence model [11], and the MRF based models [25][26]. These models have shown benefit of capturing dependencies. However, they focus on the utilization of phrases as indexing units, rather than the phrase-to-phrase semantic relationships.

The translation model-based approach proposed in [12] tries to extract phrase-to-phrase relationships according to clickthrough data. Such relationships are expected to be more effective in bridging the gap between queries and documents. In particular, the phrase translation model learns a probability distribution over "translations" of multi-word phrases from documents to queries. Assuming that queries and documents are composed using two different "languages", the phrases can be viewed as *bilingual phrases* (or *bi-phrases* in short), which are consecutive multi-term sequences that can be translated from one language to another as units. In [12], it was shown that the phrase model is more powerful than word translation models [3] because words in the relationships are considered with some context words within a phrase. Therefore, more precise translations can be determined for phrases than for words. Recent studies show that this approach is highly effective when large amounts of clickthrough data are available for training [12][15]. However, as discussed before, the phrase-based translation model can only score phrase pairs observed in the training data, and cannot generalize to new phrases. In contrast, the CLSM can generalize to model any context. In our experiments reported in Section 5, we will compare the CLSM with the word-based and phrase-based translation models.

2.2 Latent Semantic Models

The most well-known linear projection model for IR is LSA [9]. It models the whole document collection using a $n \times d$ document-term matrix $\mathbf{C}$, where n is the number of documents and d is the number of word types. $\mathbf{C}$ is first factored into the product of three matrices using singular value decomposition (SVD) as $\mathbf{C} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where the orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ are called term and document vectors, respectively, and the diagonal elements of $\mathbf{\Sigma}$ are singular values in descending order. Then, a low-rank matrix approximation of $\mathbf{C}$ is generated by retaining only the k biggest singular values in $\mathbf{\Sigma}$. Now, a document (or a query) represented by a term vector $\mathbf{D}$ can be mapped to a low-dimensional concept vector $\widehat{\mathbf{D}} = \mathbf{A}^T\mathbf{D}$, where the $d \times k$ matrix $\mathbf{A} = \mathbf{U}_k\mathbf{\Sigma}_k^{-1}$ is called the projection matrix. In document search, the relevance score between a query and a document, represented respectively by term

vectors **Q** and **D**, is assumed to be proportional to their cosine similarity score of the corresponding concept vectors $\hat{\mathbf{Q}}$ and $\hat{\mathbf{D}}$, according to the projection matrix **A**.

Generative topic models are also widely used for IR. They include Probabilistic Latent Semantic Analysis (PLSA) [19] and its extensions such as Latent Dirichlet Allocation (LDA) [4][39]. PLSA assumes that each document has a multinomial distribution over topics (called the document-topic distribution), where each of the topics is in turn of a multinomial distribution over words (called the topic-word distribution). The relevance of a query to a document is assumed to be proportional to the likelihood of generating the query by that document. Recently, topic models have been extended so that they can be trained on clickthrough data. For example, a generative model called Bi-Lingual Topic Model (BLTM) is proposed for Web search in [15], which assumes that a query and its clicked document share the same document-topic distribution. It is shown that, by learning the model on clicked query-title pairs, the BLTM gives superior performance over PLSA [15].

2.3 Neural-Network-based Semantic Models

Deep architectures have been shown to be highly effective in discovering from training data the hidden structures and features at different levels of abstraction useful for a variety of tasks [32][18][20][37][7][34]. Among them, the DSSM proposed in [20] is most relevant to our work. The DSSM uses a feed-forward neural network to map the raw term vector (i.e., with the bag-of-words representation) of a query or a document to its latent semantic vector, where the first layer, also known as the *word hashing* layer, converts the term vector to a letter-trigram vector to scale up the training. The final layer's neural activities form the vector representation in the semantic space. In document retrieval, the relevance score between a document and a query is the cosine similarity of their corresponding semantic concept vectors, as in Eq. (1). The DSSM is reported to give superior IR performance to other semantic models.

However, since the DSSM treats a query or a document as a bag of words, the fine-grained contextual structures within the query (or the document) are lost. In contrast, the CLSM is designed to capture important word n-gram level and sentence-level contextual structures that the DSSM does not. Specifically, the CLSM directly represents local contextual features at the word n-gram level; i.e., it projects each raw word n-gram to a low-dimensional feature vector where semantically similar word n-grams are projected to vectors that are close to each other in this feature space. Moreover, instead of simply summing all local word-n-gram features evenly, the CLSM performs a max pooling operation to select the highest neuron activation value across all word n-gram features at each dimension, so as to extract the sentence-level salient semantic concepts. Meanwhile, for any sequence of words, this operation forms a fixed-length sentence-level feature vector, with the same dimensionality as that of the local word n-gram features.

Deep convolutional neural networks (CNN) have been applied successfully in speech, image, and natural language processing [8][41][7]. The work presented in this paper is the first successful attempt in applying the CNN-like methods to IR. One main difference from the conventional CNN is that the convolution operation in our CLSM is applied implicitly on the letter-trigram representation space with the learned convolutional matrix W_c. The explicit convolution, with the "receptive field" of a size of three words shown in Figure 1 is accomplished by the letter-trigram matrix W_f which is fixed and not learned. Other deep learning approaches that are related to the CLSM include word-to-vector mapping (also known as word embedding) using deep neural networks learned on large amounts of raw text [1][27]. In [28], the vector representation of a word sequence is computed as a summation of embedding vectors of all words. An alternative approach is proposed in [34], where a parsing tree for a given sentence is extracted, which is then mapped to a fixed-length representation using recursive auto-encoders. Recently, a neural network based DeepMatch model is also proposed to directly capture the correspondence between two short texts without explicitly relying on semantic vector representations [23].

3. EXTRACTING CONTEXTUAL FEATURES FOR IR USING CLSM

3.1 The CLSM Architecture

The architecture of the CLSM is illustrated in Figure 1. The model contains (1) a word-n-gram layer obtained by running a contextual sliding window over the input word sequence (i.e., a query or a document), (2) a letter-trigram layer that transforms each word-trigram into a letter-trigram representation vector, (3) a convolutional layer that extracts contextual features for each word with its neighboring words defined by a window, e.g., a word-n-gram, (4) a max-pooling layer that discovers and combines salient word-n-gram features to form a fixed-length sentence-level feature vector, and (5) a semantic layer that extracts a high-level semantic feature vector for the input word sequence. In what follows, we describe these components in detail, using the annotation illustrated in Figure 1.

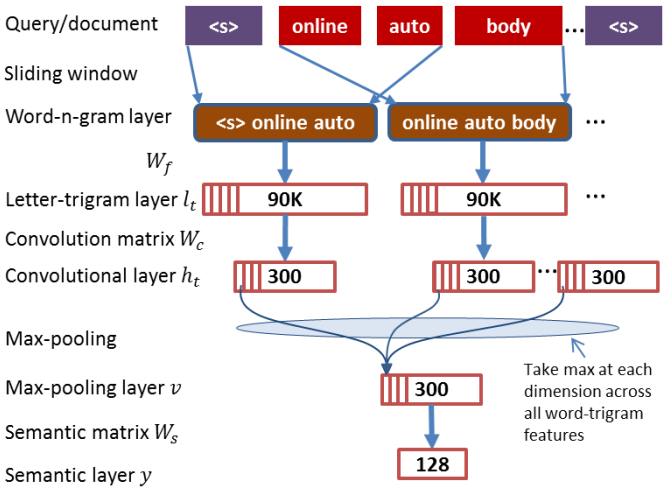

Figure 1: The CLSM maps a variable-length word sequence to a low-dimensional vector in a latent semantic space. A word contextual window size (i.e. the receptive field) of three is used in the illustration. Convolution over word sequence via learned matrix W_c is performed implicitly via the earlier layer's mapping with a local receptive field. The dimensionalities of the convolutional layer and the semantic layer are set to 300 and 128 in the illustration, respectively. The max operation across the sequence is applied for each of 300 feature dimensions separately. (Only the first dimension is shown to avoid figure clutter.)

3.2 Letter-trigram based Word-n-gram Representation

Conventionally, each word *w* is represented by a one-hot word vector where the dimensionality of the vector is the size of the vocabulary. However, the vocabulary size is often very large in real-world Web search tasks, and the one-hot vector word representation makes model learning very expensive. Therefore, we resort to a technique called *word hashing* proposed in [20], which represents a word by a letter-trigram vector. For example, given a word (e.g. *boy*), after adding word boundary symbols (e.g. *#boy#*), the word is segmented into a sequence of letter-*n*-grams (e.g. letter-tri-grams: *#-b-o*, *b-o-y*, *o-y-#*). Then, the word is represented as a count vector of letter-tri-grams. For example, the letter-trigram representation of "*boy*" is:

$$f(boy) = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 1 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

Indices of *#-b-o*, *b-o-y*, *o-y-#* in the letter-tri-gram list, respectively.

In Figure 1, the letter-trigram matrix W_f denotes the transformation from a word to its letter-trigram count vector, which requires no learning. Even though the total number of English words may grow to be extremely large, the total number of distinct letter-trigrams in English (or other similar languages) is often limited. Therefore, it can generalize to new words unseen in the training data.

Given the letter-trigram based word representation, we represent a word-n-gram by concatenating the letter-trigram vectors of each word, e.g., for the *t*-th word-n-gram at the word-n-gram layer, we have:

$$l_t = [f_{t-d}^T, \dots, f_t^T, \dots, f_{t+d}^T]^T, \qquad t = 1, \dots, T \tag{1}$$

where f_t is the letter-trigram representation of the *t*-th word, and $n = 2d + 1$ is the size of the contextual window. In our experiment, there are about 30K unique letter-trigrams observed in the training set after the data are lower-cased and punctuation-removed. Therefore, the letter-trigram layer has a dimensionality of $n \times 30K$.

3.3 Modeling Word-n-gram-Level Contextual Features at the Convolutional Layer

The convolution operation can be viewed as sliding window based feature extraction. It is designed to capture the word-n-gram contextual features. Consider the *t*-th word-n-gram, the convolution matrix projects its letter-trigram representation vector l_t to a contextual feature vector h_t. As shown in Figure 1, h_t is computed by

$$h_t = tanh(W_c \cdot l_t), \qquad t = 1, \dots, T \tag{2}$$

where W_c is the feature transformation matrix, as known as the convolution matrix, that are shared among all word n-grams. $tanh$ is used as the activation function of the neurons:

$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3}$$

The output of the convolutional layer is a variable length sequence of feature vectors, whose length is proportional to the length of the input word sequence. A special "padding" word, <s>, is added at the beginning and the end of the input word sequence so that a full window for a word at any position in the word sequence can be formed. Figure 1 shows a convolutional layer using a 3-word contextual window. Note that like the conventional CNN, the convolution matrix used in our CLSM is shared among all n-word phrases and therefore generalizes to new word-n-grams unseen in the training set.

At the convolutional layer, words within their contexts are projected to vectors that are close to each other if they are semantically similar. Table 2 presents a set of sample word-tri-grams. Considering the word "*office*" as the word of interest, we measure the cosine similarity between the contextual feature vector of "*office*" within the context "microsoft office software" and the vector of "*office*" within other contexts. We can see that the similarity scores between the learned feature vector of "microsoft office software" and those of the contexts where "*office*" is referred to the software are quite high, while the similarity scores between it and the features vectors where "*office*" has the search intent of "working space" are significantly lower. Similarly, as shown in Table 2, the context vectors of "*body*" are much closer when they are of the same search intent.

microsoft *office* software		car *body* shop	
Free *office* 2000	0.550	car *body* kits	0.698
download *office* excel	0.541	auto *body* repair	0.578
word *office* online	0.502	auto *body* parts	0.555
apartment *office* hours	0.331	wave *body* language	0.301
massachusetts *office* location	0.293	calculate *body* fat	0.220
international *office* berkeley	0.274	forcefield *body* armour	0.165

Table 2: Sample word n-grams and the cosine similarities between the learned word-n-gram feature vectors of "*office*" and "*body*" in different contexts after the CLSM is trained.

3.4 Modeling Sentence-Level Semantic Features Using Max Pooling

A sequence of local contextual feature vectors is extracted at the convolutional layer, one for each word-n-gram. These local features need to be aggregated to obtain a sentence-level feature vector with a fixed size independent of the length of the input word sequence. Since many words do not have significant influence on the semantics of the sentence, we want to suppress the non-significant local features and retain in the global feature vector only the salient features that are useful for IR. For this purpose, we use a max operation, also known as *max pooling*, to force the network to retain only the most useful local features produced by the convolutional layers. I.e., we select the highest neuron activation value across all local word n-gram feature vectors at each dimension. Referring to the max-pooling layer of Figure 1, we have

$$v(i) = \max_{t=1,\dots,T} \{h_t(i)\}, \qquad i = 1, \dots, K$$

where $v(i)$ is the *i*-th element of the max pooling layer *v*, $h_t(i)$ is the *i*-th element of the *t*-th local feature vector h_t. *K* is the dimensionality of the max pooling layer, which is the same as the dimensionality of the local contextual feature vectors $\{h_t\}$.

Table 3 shows several examples of the output of the max-pooling layer of the CLSM after training. For each sentence, we examine the five most active neurons at the max-pooling layer, measured by $v(i)$, and highlight the words in **bold** who win at these five neurons in the *max* operation (e.g., whose local features give these five highest neuron activation values)[2]. These examples show that the important concepts, as represented by these key words, make the most significant contribution to the overall semantic meaning of the sentence.

microsoft **office excel** could allow remote **code execution**
welcome to the **apartment office**
online **body fat** percentage **calculator**
online **auto body** repair **estimates**
vitamin a the **health** benefits given by **carrots**
calcium supplements and **vitamin d** discussion stop **sarcoidosis**

Table 3: Sample document titles. We examine the five most active neurons at the max-pooling layer and highlight the words in **bold** who win at these five neurons in the *max* operation. Note that, the feature of a word is extracted from that word together with the context words around it, but only the center word is highlighted in bold.

3.5 Latent Semantic Vector Representations

After the sentence-level feature vector is produced by the max-pooling operation, one more non-linear transformation layer is applied to extract the high-level semantic representation, denoted by y. As shown in Figure 1, we have

$$y = tanh(W_s \cdot v)$$

where *v* is the global feature vector after max pooling, W_s is the semantic projection matrix, and y is the vector representation of the input query (or document) in the latent semantic space, with a dimensionality of *L*.

In the current implementation of the CLSM, we use one fully-connected semantic layer, as shown in Figure 1. The model architecture can be easily extended to using more powerful, multi-layer fully-connected deep neural networks.

3.6 Using the CLSM for IR

Given a query and a set of documents to be ranked, we first compute the semantic vector representations for the query and all the documents using the CLSM as described above. Then, similar to Eq. (1), we compute the relevance score between the query and each document by measuring the cosine similarity between their semantic vectors. Formally, the semantic relevance score between a query Q and a document D is defined as:

$$R(Q,D) = \text{cosine}(y_Q, y_D) = \frac{{y_Q}^T y_D}{\|y_Q\|\|y_D\|} \quad (4)$$

[2] One word could win at multiple neurons.

where y_Q and y_D are the semantic vectors of the query and the document, respectively. In Web search, given the query, the documents are ranked by their semantic relevance scores.

4. Learning the CLSM for IR

The data for training the CLSM is the clickthrough data logged by a commercial search engine. The clickthrough data consist of a list of queries and their clicked documents, similar to the clickthrough data have been used in earlier studies, such as [12][15][20]. Similar to these work, we assume that a query is relevant to the documents that are clicked on for that query, and train the CLSM on the clickthrough data in such a way that the semantic relevance scores between the clicked documents given the queries are maximized.

Following [20], we first convert the semantic relevance score between a query and a *positive* document to the posterior probability of that document given the query through softmax:

$$P(D^+|Q) = \frac{\exp(\gamma R(Q,D^+))}{\sum_{D' \in \boldsymbol{D}} \exp(\gamma R(Q,D'))} \quad (5)$$

where γ is a smoothing factor in the softmax function, which is set empirically on a held-out data set in our experiment. $\boldsymbol{D}$ denotes the set of candidate documents to be ranked. In practice, for each (query, clicked-document) pair, we denote by (Q, D^+) where Q is a query and D^+ is the clicked document and approximate $\boldsymbol{D}$ by including D^+ and J randomly selected unclicked documents, denote by $\{D_j^-; j = 1, \dots, J\}$. As an alternative, the model could also be trained using noise contrastive estimation as in [29].

In training, the model parameters are learned to maximize the likelihood of the clicked documents given the queries across the training set. That is, we minimize the following loss function

$$L(\Lambda) = -\log \prod_{(Q,D^+)} P(D^+|Q) \quad (6)$$

where Λ denotes the parameter set of the CLSM.

Note that the loss function of Eq. (5) and (6) covers the pair-wise loss that has been widely used for learning-to-rank [5] as a special case if we allow only one unclicked document to be sampled. This loss function is also widely used in speech recognition and other applications [17]. It is more flexible than pairwise loss in exploring different sampling techniques for generating unclicked documents for discriminative information.

To determine the training hyper parameters and to avoid over-fitting, we divide the clickthrough data into two sets that do not overlap, called training and validation datasets, respectively. In our experiments, the models are trained on the training data and the training parameters are optimized on the validation data.

The weights of the neural network are randomly initialized as suggested in [30]. The model is trained using mini-batch based stochastic gradient descent. Each mini-batch consists of 1024 training samples. In our implementation, models are trained using an NVidia Tesla K20 GPU.

5. EXPERIMENTS

5.1 Data Sets and Evaluation Methodology

We evaluated the retrieval models on a large-scale, real-world data set, called the evaluation data set henceforth. The evaluation data set contains 12,071 English queries sampled from one-year

query log files of a commercial search engine and labeled by human judgers. On average, each query is associated with 74 Web documents (URLs). Each query-document pair has a relevance label manually annotated on a 5-level relevance scale: *bad*, *fair*, *good*, *excellent*, and *perfect*, corresponding to 0 to 4, where level 0 (*bad*) means D is not relevant to Q and level 4 (*perfect*) means that the document is the most relevant to query Q. All the queries and documents are preprocessed such that the text is white-space tokenized and lowercased, numbers are retained, and no stemming/inflection is performed. Figure 2 shows the length distributions of queries and documents in the evaluation data. The average lengths of the queries and the document titles are 3.01 and 7.78 words, respectively.

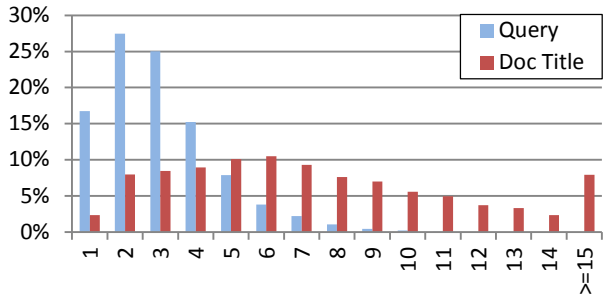

Figure 2: The distribution of query length and document title length in the evaluation data set. The evaluation set consists of 12,071 queries and 897,770 documents. Query length is 3.01 on average. Document title length is 7.78 on average.

As mentioned earlier, we have used only the title field of a Web document for ranking in our experiments. As shown in Table 4, the title field is very effective for document retrieval, although titles are much shorter than body texts.

Field	NDCG@1	NDCG@3	NDCG@10
Body	0.275	0.310	0.375
Title	0.305^{α}	0.328^{α}	0.388^{α}

Table 4: Ranking results of two BM25 models, each uses a different single field to represent Web documents. The superscript α indicates statistically significant improvements ($p < 0.05$) over **Body**.

All the ranking models used in this study contain many free hyper-parameters that must be estimated empirically. In all experiments, we have used 2-fold cross validation: A set of results on one half of the data is obtained using the parameter settings optimized on the other half, and the global retrieval results are combined from those of the two sets.

The performance of all ranking models we have evaluated has been measured by mean Normalized Discounted Cumulative Gain (NDCG) [21], and we will report NDCG scores at truncation levels 1, 3, and 10. We have also performed a significance test using the paired *t*-test. Differences are considered statistically significant when the *p*-value is lower than 0.05.

In our experiments, the clickthrough data used for model training include 30 million query and clicked-title pairs sampled from one year query log files. The query-title pairs are pre-processed in the same way as the evaluation data to ensure uniformity. We test the models in ranking the documents in the evaluation data set. There is no overlap between the training set and the evaluation set.

5.2 Model Settings and Baseline Performance

We have compared the CLSM with five sets of baseline models, as shown in Table 5. The first set includes two widely used lexical matching methods, BM25 and the unigram language model (ULM). The second set includes a set of state-of-the-art latent semantic models which are learned either on documents only in an unsupervised manner (PLSA and LDA) or on clickthrough data in a supervised way (BLTM). The third set includes a phrase-based translation model (PTM) that intends to directly model the contextual information within a multi-term phrase. This set also includes a word-based translation model (WTM) which is a special case of the phrase-based translation model. Both translation models are learned on the same clickthrough data described in Section 5.1. The fourth set includes the MRF based term-dependency model and the latent concept expansion (LCE) model. The fifth set includes the DSSM, which is a deep neural network based model, which is also learned on the same clickthrough data. In order to make the results comparable, we re-implement these models following the descriptions in [11][15][20][25][26]. Details are elaborated in the following paragraphs.

BM25 and ULM are used as baselines. Both models use the term vector representation for queries and documents. BM25 (Row 1 in Table 5) follows the BM25 term weighting function used in the Okapi system. ULM (Row 2) is a unigram language model with Dirichlet smoothing [42]. Both ULM and BM25 are state-of-the-art document ranking models based on term matching. They have been widely used as baselines in related studies.

PLSA (Rows 3 and 4) is our implementation of the model proposed in [19], and was trained on documents only (i.e., the title side of the query/clicked-title pairs). Different from [19], our version of PLSA was learned using MAP estimation as in [15]. We experimented with different numbers of topics, and the results of using 100 topics and 500 topics are reported in Row 3 and 4, respectively. In our experiments, they give similar performance.

LDA (Row 5 and 6) is our implementation of the model in [39]. It was trained on documents only (i.e., the title side of the query/clicked-title pairs). The LDA model is learned via Gibbs sampling. The number of topics is set to 100 and 500, respectively. LDA gives slightly better results than the PLSA, and LDA with 500 topics significantly outperforms BM25 and ULM.

BLTM (Row 7) is the best performer among different versions of the bilingual topic models described in [15]. It is trained on query-title pairs using the EM algorithm with a constraint enforcing the paired query and title to have same fractions of terms assigned to each hidden topic. The number of topics is set to 100 as in [15]. We see that using clickthrough data for model training leads to improvement over PLSA and LDA. BLTM also significantly outperforms BM25 and ULM.

MRF (Row 8) models the term dependency using a MRF as described in [25]. We use cross-validation method to tune the optimal parameters for the feature weights.

LCE (Row 9) is a latent concept expansion model proposed in [26]. It leverages the term-dependent information by adding n-gram and (unordered n-gram) as features into the log-linear ranking model. In our experiments, we re-implemented LCE following [26]. Both MRF and LCE outperform BM25 and ULM significantly.

WTM (Row 10) is our implementation of the word-based translation model described in [12], which is a special case of the phrase-based translation model, listed here for comparison.

PTM (Row 11) is the phrase-based translation model proposed in [12]. It is supposed to be more powerful than WTM because words in the relationships are considered with contextual words within a phrase. Therefore, more precise translations can be determined for phrases than for words. The model is trained on query-title pairs. The maximum length of a phrase is set to three. Our results are consistent with that of [11], showing that phrase models are more effective for retrieval than word models when large amounts of clickthrough data are available for training.

DSSM (Row 12 and 13) is the best variant of DSSM proposed in [20]. It includes the letter-trigram based word hashing layer, two non-linear hidden layers, each of which has 300 neurons, and an output layer that has 128 neurons. In our experiments, we found that learning two separate neural networks, one for the query and one for the document title, gives better performance than sharing the same neural network for both of the query and the document title. Therefore, we always use two separate neural networks in our experiments thereafter. We have experimented with using different number of negative samples, J, in the training of the DSSM. Row 12 uses the setting of $= 4$, where Row uses the setting of $J = 50$. The DSSMs are trained on the same query-title pairs described in section 5.1 [3]. The results in Table 5 confirm that the DSSM (e.g., Row 13) outperforms other competing models in Rows 1 to 11 significantly. The results also show that using more negative samples in training leads to better results (e.g., Row 13 vs. Row 12).

CLSM (Row 14 and 15) is the proposed CLSM described in Sections 3 and 4. The convolutional layer and max-pooling layer each has 300 neurons, and the final output layer has 128 neurons. Two separate convolutional neural networks are used in the experiments. We have also experimented with using different number of negative samples, J, in the training of the CLSM. The model is trained on the same query-title clickthrough dataset described in section 5.1.

5.3 Results

The main results of our experiments are summarized in Table 5. First, we observe that the CLSM ($J = 50$) outperforms the state-of-the-art term matching based document ranking models, BM25 and ULM, with a substantial margin of 4.3% in NDCG@1. The CLSM also outperforms the state-of-the-art topic model based approaches (i.e., PLSA, LDA, and BLTM) with a statistically significant margin from 3.2% to 4.0%. Further, compared to previous term-dependency models, the CLSM with the best setting outperforms MRF, LCE and PTM by a substantial improvement of 3.3%, 3.6%, and 2.9% NDCG@1 respectively. This demonstrates CLSM's effectiveness in capturing the contextual structure useful for semantic matching. Finally, we obtain significant 2.2% to 2.3% NDCG@1 improvement of the CLSM over DSSM, a state-of-the-art neural network based model. This demonstrates the importance of CLSM's capability of modeling fine-grained word n-gram level and sentence-level contextual structures for IR, as the DSSM is based on the bag-of-words representation and cannot capture such information.

We then further investigated the performance of CLSM using different context window sizes and present the experimental results in Table 6. In Table 6, we first observe that even with a context window size of one, the CLSM still significantly outperforms the DSSM, demonstrating that it is far more effective for IR to capture salient local features than simply summing over the contributions from all words uniformly. Then, when increasing the window size from one to three, we observe another significant improvement, attributes to the capability of modeling word-tri-gram contextual information. When the window size is increased to five, however, no significant gain is observed. Our interpretation is that because the average lengths of the queries and the document titles are only three and eight words respectively, window sizes larger than three do not provide much extra context information. Moreover, big context windows lead to more model parameters to learn, and thus increase the difficulty of parameter learning. In the next subsection, we will present an in-depth analysis on the performance of the CLSM.

#	Models	NDCG@1	NDCG@3	NDCG@10
1	BM25	0.305	0.328	0.388
2	ULM	0.304	0.327	0.385
3	PLSA (T=100)	0.305	0.335 $^{\alpha}$	0.402 $^{\alpha}$
4	PLSA (T=500)	0.308	0.337 $^{\alpha}$	0.402 $^{\alpha}$
5	LDA (T=100)	0.308	0.339 $^{\alpha}$	0.403 $^{\alpha}$
6	LDA (T=500)	0.310 $^{\alpha}$	0.339 $^{\alpha}$	0.405 $^{\alpha}$
7	BLTM	0.316 $^{\alpha}$	0.344 $^{\alpha}$	0.410 $^{\alpha}$
8	MRF	0.315 $^{\alpha}$	0.341 $^{\alpha}$	0.409 $^{\alpha}$
9	LCE	0.312 $^{\alpha}$	0.337 $^{\alpha}$	0.407 $^{\alpha}$
10	WTM	0.315 $^{\alpha}$	0.342 $^{\alpha}$	0.411 $^{\alpha}$
11	PTM (maxlen = 3)	0.319 $^{\alpha}$	0.347 $^{\alpha}$	0.413 $^{\alpha}$
12	DSSM ($J = 4$)	0.320 $^{\alpha}$	0.355 $^{\alpha\beta}$	0.431 $^{\alpha\beta}$
13	DSSM ($J = 50$)	0.327 $^{\alpha\beta}$	0.363 $^{\alpha\beta}$	0.438 $^{\alpha\beta}$
14	CLSM ($J = 4$)	0.342 $^{\alpha\beta\gamma}$	0.374 $^{\alpha\beta\gamma}$	0.447 $^{\alpha\beta\gamma}$
15	**CLSM ($J = 50$)**	**0.348** $^{\alpha\beta\gamma}$	**0.379** $^{\alpha\beta\gamma}$	**0.449** $^{\alpha\beta\gamma}$

Table 5: Comparative results with the previous state of the art approaches. BLTM, WTM, PTM, DSSM, and CLSM use the same clickthrough data described in section 5.1 for learning. Superscripts $\alpha, \beta,$ and γ indicate statistically significant improvements ($p < 0.05$) over **BM25**, **PTM**, and **DSSM ($J = 50$)**, respectively.

#	Models	NDCG @1	NDCG @3	NDCG @10
1	DSSM ($J = 50$)	0.327	0.363	0.438
2	CLSM ($J = 50$) win =1	0.340 $^{\alpha}$	0.374 $^{\alpha}$	0.443 $^{\alpha}$
3	CLSM ($J = 50$) win =3	0.348 $^{\alpha\beta}$	0.379 $^{\alpha\beta}$	0.449 $^{\alpha\beta}$
4	CLSM ($J = 50$) win =5	0.344 $^{\alpha}$	0.376 $^{\alpha}$	0.448 $^{\alpha\beta}$

Table 6: Comparative results of the CLSMs using different convolution window sizes. The setting of the DSSM is 300/300/128 and the setting of the CLSM is K=300, L=128. Superscripts α and β indicate statistically significant improvements ($p < 0.05$) over DSSM and CLSM (win=1), respectively.

5.4 Analysis

In order to gain a better understanding of the difference between models, we compare the CLSM with the DSSM query by query under their best settings, i.e., row 13 and row 15 in table 5, respectively.

[3] For comparison, we re-implemented the DSSM on the current data set. The data set used in [20] is encoded in a bag-of-words representation format and thus not suitable for this study (personal communication).

Query	Title of the top-1 returned document retrieved by CLSM
warm environment arterioles do what	**thermoregulation** wikipedia the free encyclopedia
auto body repair cost **calculator** software	free online **car body shop** repair **estimates**
what happens if our body **absorbs excessive** amount **vitamin d**	**calcium** supplements and **vitamin d** discussion stop **sarcoidosis**
how do **camera** use **ultrasound focus** automatically	**wikianswers** how does a **camera focus**
how to **change font excel office 2013**	**change font** default styles in **excel 2013**
12 **fishing boats trailers**	**trailer kits** and accessories **motorcycle** utility **boat snowmobile**
acp **ariakon combat pistol** 2.0	paintball acp **combat** pistol paintball **gun paintball pistol** package deal marker and gun

Table 8: Samples of queries and the top-1 documents ranked by the CLSM. Words marked in **bold** are those that contribute to the five most active neurons at the max-pooling layer.

For each query, we checked the relevance label of the top-1 document retrieved by the DSSM and the CLSM, respectively. We count the number of times that the retrieved document is in each of the five relevance category, from *bad* to *perfect*, for both models. The distributions of the relevance labels of the top-1 returned documents are plotted in Figure 3. Compared with the DSSM, overall, the CLSM returns more relevant documents, i.e., the percentages of returned documents in the *bad* or *fair* categories decrease substantially, and the percentage of returned documents in the *good* category increases substantially. The counts in the *excellent* and *perfect* categories also increase, although the absolute numbers are small.

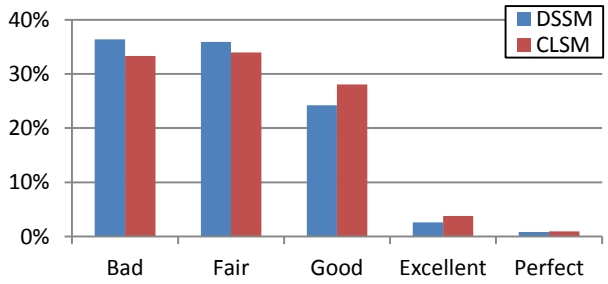

Figure 3: Percentage of top-1 ranked documents in each of the five relevance categories, retrieved by the CLSM and the DSSM, respectively.

Models		DSSM		
	Label	*bad*	*fair*	*good+*
CLSM	*bad*	3052	626	342
	fair	738	2730	631
	good+	601	981	2370

Table 7: CLSM vs DSSM on Top-1 search results. The three relevance categories, *good*, *excellent*, and *perfect*, are merged into one *good+* category.

A more detailed comparison between the CLSM and the DSSM is presented in table 7. We observe that for 8,152 of the total 12,071 queries, both the CLSM and the DSSM return the documents of the same quality. However, in the cases where they return documents with different qualities, the advantage of the CLSM over the DSSM can be clearly observed. For example, there are 601 queries for which the CLSM returns *good* or better quality Top-1 documents while the DSSM's Top-1 returns are *bad*, much more than the opposite cases. There are also 981 queries for which the CLSM returns *good* or better Top-1 documents while the DSSM returns *fair* documents, much more than the 631 opposite cases.

To help better understand what is learned by the CLSM, we show several examples selected from the CLSM result on the evaluation data set in Table 8. Each row includes a query and the title of the top 1 document ranked by CLSM. In both of the query and the document title, the words that most significantly contribute to the semantic meaning, e.g., words contribute to the most active five neurons at the max pooling layer, are marked in **bold**. To further illustrate the CLSM's capability for semantic matching, we trace the activation of neurons at the max-pooling layer for the first three examples in Table 8 and elaborate these examples in Figure 4. We first project both the query and the document title to the max-pooling layer, respectively. Then, we evaluate the activation values of neurons at the max-pooling layer, and show the indices of the neurons that have high activation values for both query and document title, e.g., the product of the activation values of the query and the document title at a neuron is larger than a threshold. After that, we trace back to the words that win these neurons in both the query and the document title. In Figure 4, we show the indices of these matching neurons and the words in the query and the document title that win them.

In the first example, though there is no overlap between the key words "warm environment arterioles" in the query and the word "thermoregulation" in the document, they both have high activation values at a similar set of neurons, and thus lead to a query-document match in the semantic space. Similar behavior is observed in the second example. "auto" and "calculator" in the query and "car" and "estimates" in the document activate similar neurons, thus leading to a query-document match in the semantic space as well. The third example is more complicated. "vitamin d" is closely associated to "calcium absorbing", and "excessive calcium absorbing" is a symptom of "sarcoidosis". In Figure 4 (c), we observe that both "calcium" in the document title and "d" (with its context "vitamin") in the query gives high activation at neuron 88, while "sarcoidosis" in the document title and "absorbs" "excessive" and "vitamin" in the query have high activations at the set of neurons 90, 66, 79. Our analysis indicates that different words with related semantic meanings activate the similar set of neurons, resulting to a high overall matching score. This demonstrates the effectiveness of the CLSM in extracting the salient semantic meaning in queries and documents for Web search.

6. SUMMARY

In this paper, we have reported a novel deep learning architecture called the CLSM, motivated by the convolutional structure of the CNN, to extract both local contextual features at the word-n-gram level (via the convolutional layer) and global contextual features at the sentence-level (via the max-pooling layer) from text. The higher layer(s) in the overall deep architecture makes effective use

of the extracted context-sensitive features to generate latent semantic vector representations which facilitates semantic matching between documents and queries for Web search applications. We have carried out extensive experimental studies of the proposed model whereby several state-of-the-art semantic models are compared and significant performance improvement on a large-scale real-world Web search data set is observed.

Extended from our previous work [20] [33], the CLSM and its variations have also been demonstrated giving superior performance on a range of natural language processing tasks beyond information retrieval, including machine translation [13], semantic parsing and question answering [40], entity search and online recommendation [14]. In the future, the CLSM can be further extended to automatically capture a wider variety of types of contextual features from text than our current settings.

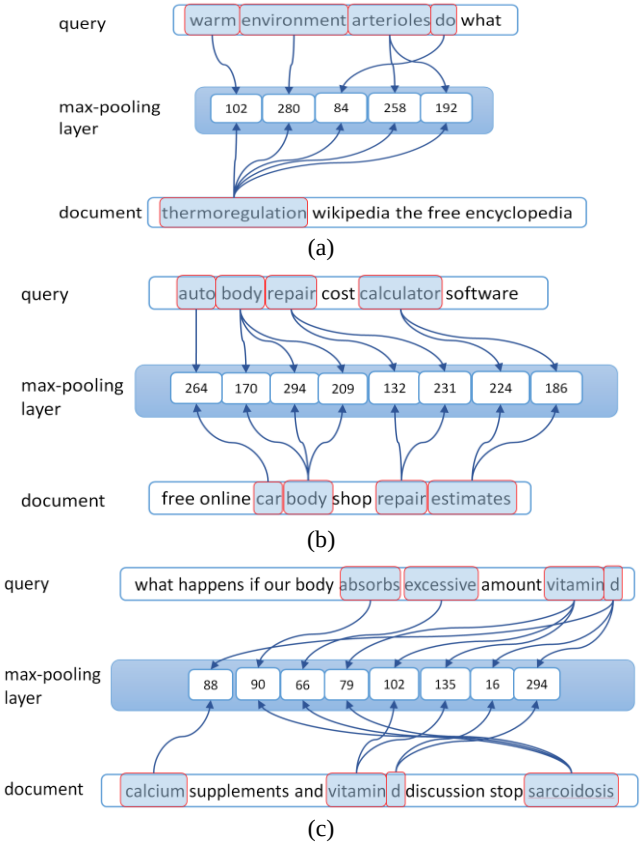

Figure 4: Illustration of semantic matching between a query and a document title at the max-pooling layer, after word-n-gram contextual feature extraction and the max pooling operation. The indices of the neurons at the max-pooling layer that have high activation values for both query and document title are shown.

REFERENCES

[1] Bengio, Y., 2009. Learning deep architectures for AI. In Foundamental Trends in Machine Learning, vol. 2, no. 1.

[2] Bendersky, M., Metzler, D., and Croft, B., 2011. Parameterized concept weighting in verbose queries. In SIGIR, pp. 605-614.

[3] Berger, A., and Lafferty, J. 1999. Information retrieval as statistical translation. In SIGIR, pp. 222-229.

[4] Blei, D. M., Ng, A. Y., and Jordan, M. J. 2003. Latent Dirichlet allocation. In Journal of Machine Learning Research, 3: 993-1022.

[5] Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, and Hullender, G. 2005. Learning to rank using gradient descent. In ICML, pp. 89-96.

[6] Buckley, D., Allan, J., and Salton, G. 1995. Automatic retrieval approaches using SMART: TREC-2. Information Processing and Management, 31: 315-326.

[7] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P., 2011. Natural language processing (almost) from scratch. In Journal of Machine Learning Research, vol. 12.

[8] Deng, L., Abdel-Hamid, O., and Yu, D., 2013. A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion, in ICASSP.

[9] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T., and Harshman, R. 1990. Indexing by latent semantic analysis. Journal of the American Society for Information Science, 41(6): 391-407.

[10] Dumais, S. T., Letsche, T. A., Littman, M. L., and Landauer, T. K. 1997. Automatic cross-linguistic information retrieval using latent semantic indexing. In AAAI-97 Spring Symposium Series: Cross-Language Text and Speech Retrieval.

[11] Gao, J., Nie, J-Y., Wu, G. and Cao, G. 2004. Dependence language model for information retrieval. In SIGIR.

[12] Gao, J., He, X., and Nie, J-Y. 2010. Clickthrough-based translation models for web search: from word models to phrase models. In CIKM, pp. 1139-1148.

[13] Gao, J., He, X., Yih, W-T., and Deng, L., 2014. Learning continuous phrase representations for translation modeling. In ACL.

[14] Gao, J., Pantel, P., Gamon, M., He, X., Deng, L., and Shen, Y. 2014. Modeling interestingness with deep neural networks. In EMNLP.

[15] Gao, J., Toutanova, K., Yih., W-T. 2011. Clickthrough-based latent semantic models for web search. In SIGIR, pp. 675-684.

[16] Girolami, M., and Kaban, A. 2003. On an equivalence between PLSA and LDA. In SIGIR, pp. 433-434.

[17] He, X., Deng, L., and Chou, W., 2008. Discriminative learning in sequential pattern recognition. In IEEE Signal Processing Magazine. vol 5.

[18] Hinton, G., and Salakhutdinov, R., 2010. Discovering binary codes for documents by learning deep generative models. In Topics in Cognitive Science, pp 1-18.

[19] Hofmann, T. 1999. Probabilistic latent semantic indexing. In SIGIR, pp. 50-57.

[20] Huang, P., He, X., Gao, J., Deng, L., Acero, A., and Heck, L. 2013. Learning deep structured semantic models for web search using clickthrough data. In CIKM.

[21] Jarvelin, K. and Kekalainen, J. 2000. IR evaluation methods for retrieving highly relevant documents. In SIGIR.

[22] Lavrenko, V., and Croft, B., 2001. Relevance-based language models. In SIGIR, pp. 120-127.

[23] Lu, Z. and Li, H. 2013. A deep architecture for matching short texts. In NIPS.

[24] Lv, Y., and Zhai, C., 2009. Positional Language Models for Information Retrieval. In SIGIR, pp. 299-306.

[25] Metzler, D. and Croft, B. 2005. A Markov random field model for term dependencies. In SIGIR.
[26] Metzler, D., and Croft, B. 2007. Latent Concept Expansion using Markov Random Fields. In SIGIR, pp. 311-318.
[27] Mikolov, T., Yih, W., and Zweig, G., 2013. Linguistic regularities in continuous space word representations. In NAACL HLT.
[28] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In NIPS.
[29] Mnih A., and Kavukcuoglu, K., 2013. Learning word embeddings efficiently with noise-contrastive estimation. In NIPS.
[30] Montavon, G., Orr, G., Müller, K., 2012. Neural Networks: Tricks of the Trade (Second edition). Springer.
[31] Platt, J., Toutanova, K., and Yih, W. 2010. Translingual document representations from discriminative projections. In EMNLP, pp. 251-261.
[32] Salakhutdinov R., and Hinton, G., 2007. Semantic hashing. in Proc. SIGIR Workshop Information Retrieval and Applications of Graphical Models.
[33] Shen, Y., He, X., Gao, J., Deng, L., and Mesnil, G., 2014. Learning semantic representations using convolutional neural networks for web search. In WWW.
[34] Socher, R., Huval, B., Manning, C., Ng, A., 2012. Semantic compositionality through recursive matrix-vector spaces. In EMNLP.
[35] Song, F. and Croft, B. 1999. A general language model for information retrieval. In CIKM, pp. 316-321.
[36] Sparck-Jones, K. 1998. What is the role of NLP in text retrieval? In: Natural language information retrieval (Ed. T. Strzalkowski), Dordrecht: Kluwer.
[37] Tur, G., Deng, L., Hakkani-Tur, D., and He, X., 2012. Towards deeper understanding deep convex networks for semantic utterance classification. In ICASSP
[38] Wang, K., Li, X., and Gao, J. 2010. Multi-style language model for web scale information retrieval. In SIGIR.
[39] Wei, X., and Croft, W. B. 2006. LDA-based document models for ad-hoc retrieval. In SIGIR, pp. 178-185.
[40] Yih, W-T., He, X., and Meek, C., 2014. Semantic parsing for single-relation question answering. In ACL.
[41] Zeiler, M., Taylor, G., and Fergus, R., 2011. Adaptive deconvolutional networks for mid and high level feature learning. In ICCV.
[42] Zhai, C. and Lafferty, J. 2001. A study of smoothing methods for language models applied to ad hoc information retrieval. In SIGIR, pp. 334-342.

APPENDIX

The CLSM is trained using stochastic gradient descent. Let $loss(\Lambda)$ be a sample-wise loss function, the model parameters are updated by

$$\Lambda_t = \Lambda_{t-1} - \epsilon_t \frac{\partial loss(\Lambda)}{\partial \Lambda}|_{\Lambda=\Lambda_{t-1}} \quad (7)$$

where ϵ_t is the learning rate at the t^{th} iteration, Λ_t and Λ_{t-1} are the model parameters at the t^{th} and the $(t-1)^{th}$ iterations, respectively.

For each query Q, we denote by D^+ the clicked document, and $\{D_j^-; j = 1, \dots, 4\}$ the unclicked document. In the following, referring to Figure 1 and Eq. (5) – (7) for definitions of variables, we then derive $\frac{\partial loss(\Lambda)}{\partial \Lambda}$ as follows.

First, the loss function in Eq. (7) can be written as:

$$loss(\Lambda) = \log\left(1 + \sum_j \exp(-\gamma\, \Delta_j)\right) \quad (8)$$

where $\Delta_j = R(Q, D^+) - R(Q, D_j^-)$.

The gradient of the loss function w.r.t. the semantic projection matrix W_s is

$$\frac{\partial loss(\Lambda)}{\partial W_s} = \sum_j \alpha_j \frac{\partial \Delta_j}{\partial W_s} \quad (9)$$

where

$$\frac{\partial \Delta_j}{\partial W_s} = \frac{\partial R(Q, D^+)}{\partial W_s} - \frac{\partial R(Q, D_j^-)}{\partial W_s} \quad (10)$$

and

$$\alpha_j = \frac{-\gamma \exp(-\gamma\, \Delta_j)}{1 + \sum_{j'} \exp(-\gamma\, \Delta_{j'})} \quad (11)$$

To simplify the notation, let a, b, c be ${y_Q}^T y_D$, $1/\|y_Q\|$, and $1/\|y_D\|$, respectively. With $tanh$ as the activation function in our model, each term in the right-hand side of Eq. (10) can be calculated using the following formula:

$$\frac{\partial R(Q,D)}{\partial W_s} = \frac{\partial}{\partial W_s}\frac{{y_Q}^T y_D}{\|y_Q\|\|y_D\|} = \delta_{y_Q}^{(Q,D)} v_Q^T + \delta_{y_D}^{(Q,D)} v_D^T \quad (12)$$

where $\delta_{y_Q}^{(Q,D)}$ and $\delta_{y_D}^{(Q,D)}$ for a pair of (Q, D) are computed as

$$\begin{aligned} \delta_{y_Q}^{(Q,D)} &= (1 - y_Q) \circ (1 + y_Q) \circ (bcy_D - acb^3 y_Q) \\ \delta_{y_D}^{(Q,D)} &= (1 - y_D) \circ (1 + y_D) \circ (bcy_Q - abc^3 y_D) \end{aligned} \quad (13)$$

where the operator $\circ$ is the element-wise multiplication (Hadamard product).

In order to compute the gradient of the loss function w.r.t. the convolution matrix, W_c, first we also need to calculate $\{\delta\}$ for each Δ_j at the max layer. For example, each δ in the max layer, v, can be calculated through back propagation as

$$\begin{aligned} \delta_{v_Q}^{(Q,D)} &= (1 + v_Q) \circ (1 - v_Q) \circ W_s^T \delta_{y_Q}^{(Q,D)} \\ \delta_{v_D}^{(Q,D)} &= (1 + v_D) \circ (1 - v_D) \circ W_s^T \delta_{y_D}^{(Q,D)} \end{aligned} \quad (14)$$

Then we need to trace back to the local features that win in the max operation, i.e.,

$$\begin{aligned} t_Q(i) &= \underset{t=1,\dots,T_Q}{\operatorname{argmax}}\{h_t(i)\}, \qquad i = 1, \dots, K \\ t_D(i) &= \underset{t=1,\dots,T_D}{\operatorname{argmax}}\{h_t(i)\}, \qquad i = 1, \dots, K \end{aligned} \quad (15)$$

Then, the gradient of the loss function w.r.t. the convolution matrix, W_c, can be computed as

$$\frac{\partial loss(\Lambda)}{\partial W_c} = \sum_j \alpha_j \frac{\partial \Delta_j}{\partial W_c} \quad (16)$$

where for the i-th row of W_c, e.g., $W_{c,i}$, $i = 1, \dots, K$, we have:

$$\begin{aligned} \frac{\partial \Delta_j}{\partial W_{c,i}} = &\left(\delta_{v_Q}^{(Q,D^+)}(i) l_{Q,t_Q(i)}^T + \delta_{v_{D^+}}^{(Q,D^+)}(i) l_{D^+,t_{D^+}(i)}^T\right) \\ &- \left(\delta_{v_Q}^{(Q,D_j^-)}(i) l_{Q,t_Q(i)}^T + \delta_{v_{D_j^-}}^{(Q,D_j^-)}(i) l_{D_j^-,t_{D_j^-}(i)}^T\right) \end{aligned}$$

where $\delta(i)$ is the i-th element of δ, and $l_{Q,t}$ and $l_{D,t}$ is the context window vector at the t-th position of Q or D, as defined in section 3.3, respectively.