Learning to Rewrite Queries

Yunlong He
Yahoo! Reseach
Sunnyvale, CA
yunlong.he@yahoo.com

Jiliang Tang
Michigan State University
East Lansing, MI
jiliang.tang@cse.msu.edu

Hua Ouyang
Yahoo! Reseach
Sunnyvale, CA
houyang@yahoo-inc.com

Changsung Kang
Yahoo! Reseach
Sunnyvale, CA
ckang@yahoo-inc.com

Dawei Yin
Yahoo! Reseach
Sunnyvale, CA
daweiy@yahoo-inc.com

Yi Chang
Yahoo! Reseach
Sunnyvale, CA
yichang@yahoo-inc.com

ABSTRACT

It is widely known that there exists a semantic gap between web documents and user queries and bridging this gap is crucial to advance information retrieval systems. The task of query rewriting, aiming to alter a given query to a rewrite query that can close the gap and improve information retrieval performance, has attracted increasing attention in recent years. However, the majority of existing query rewriters are not designed to boost search performance and consequently their rewrite queries could be sub-optimal. In this paper, we propose a learning to rewrite framework that consists of a candidate generating phase and a candidate ranking phase. The candidate generating phase provides us the flexibility to reuse most of existing query rewriters; while the candidate ranking phase allows us to explicitly optimize search relevance. Experimental results on a commercial search engine demonstrate the effectiveness of the proposed framework. Further experiments are conducted to understand the important components of the proposed framework.

1 Introduction

Users of the Internet typically play two roles in commercial search engines. They are information creators that generate web documents and they are also information consumers that retrieve documents for their information needs. It is well known, however, that there is a "lexical chasm" [18] between user queries and web documents because they use different language styles and vocabularies. As a consequence, search engines could be unable to retrieve relevant documents even when the issued queries are perfectly describing users' information needs. For example, a user forms a query "how much tesla", but web documents in search engines use the expression "price tesla". Therefore it has become increasingly important for search engines to intelligently satisfy users' information needs by understanding the intent from their queries.

Query rewriting (QRW), which targets to alter a given query to alternative queries that can improve relevance performance by reducing the mismatches, is a critical task in modern search engines and has attracted increasing attention in recent years [6, 14, 18, 8]. For example, authors in [14] propose to modify the search queries based on typical substitutions that web searchers make to their queries; and query rewriting is treated as a machine translation problem and statistical machine translation (SMT) models are trained based on query-snippet pairs [18, 8]. However, the vast majority of existing algorithms focus on either correlations between queries and rewrites or bridging the language gap between user queries and web documents, which could be sub-optimal for the goal of query rewriting – improving search relevance performance. For example, since the SMT model aims at translating a sentence from a source language to a fluent and grammatically correct sentence in a target language, the SMT model prefers to rewrite the query "how much tesla" as "how much is the tesla" instead of "price tesla". Therefore it is necessary to explicitly consider ranking relevance when developing query rewriting methods.

In this paper, we propose a learning to rewrite framework that consists of (1) candidate generating and (2) candidate ranking as shown in Figure 1. Given a query, we create possible candidates via a set of candidate generators in the candidate generating phase; while given a learning target, we train a scoring function to rank these candidates from the candidate generating phase in the candidate ranking phase. The advantages of this framework are two-fold. First, the candidate generating phase not only allows us to reuse most of existing query rewriting algorithms as candidate generators but also enables us to explore recent advanced techniques such as deep learning to build new candidate generators. Second, the candidate ranking phase gives us flexibility to choose the target to optimize for query rewriting; for example, in this work, we desire to boost relevance performance thus we can choose the relevance target to rank candidates. Our contributions are summarized as below:

- Define the problem of learning to rewrite formally;
- Propose a learning to rewrite framework that is flexible to incorporate existing query rewriting algorithms and optimize the relevance performance explicitly; and
- Conduct experiments on a commercial search engine to

CIKM'16 , October 24-28, 2016, Indianapolis, IN, USA

DOI: http://dx.doi.org/10.1145/2983323.2983835

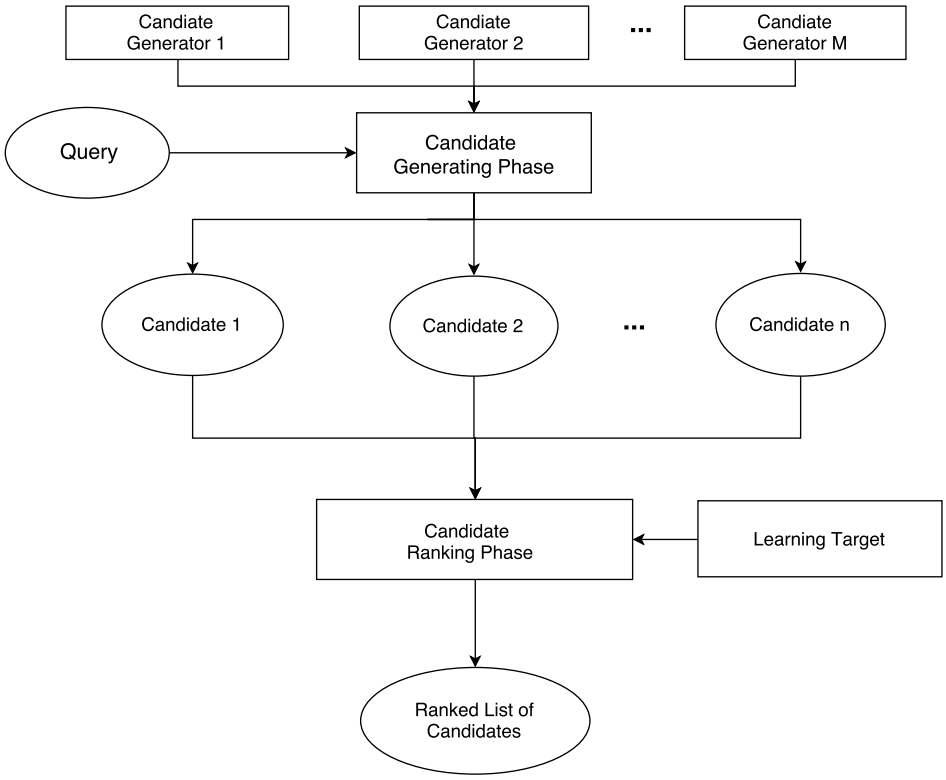

Figure 1: Flow chart of the proposed learning to rewrite framework

demonstrate the effectiveness of the proposed framework.

The rest of the paper is organized as follows. In Section 2, we briefly review the related work. In Section 3, we formally define the problem and introduce the proposed framework in detail. In Section 4, we present empirical evaluation with discussion. In Section 5, we give conclusion with future work.

2 Related Work

Query expansion and rewriting have long been important research topics in information retrieval [3]. Xu and Croft [23] studied using the top ranked documents retrieved by the original query to expand the query. This method suffers from the sensitivity to initial ranking results and does not learn from user generated data. Later approaches [6, 7, 14] focus on using user query logs to generate query expansions by collecting signals such as clickthrough rate [6, 24], cooccurrence in search sessions [14] or query similarity based on click graphs [7, 1]. Since search logs contain query document pairs clicked by millions of users, the term correlations reflect the preference of the majority of users. However, the correlation-based method, as pointed out by [19], suffers low precision partly because the correlation model does not explicitly capture context information and is susceptible to noise. More recently, natural language technology in form of statistical machine translation (SMT) [19, 18, 8, 9] has been introduced for the query expansion and rewriting problems. In the SMT system all component models are properly smoothed using sophisticated techniques to avoid sparse data problems while the correlation model relies on pure counts of term frequencies. However, the SMT system is used as a black box instead of fully tuned for the task of query rewriting.

Another related research topic is query recommendation. Authors in [12] make use of search session data to find query pairs frequently co-occurring in the same sessions, which are used as suggestions for each other. Baeza-Yates et al. [2] propose to rank the clustered queries according to two principles: i) the similarity of the queries to the input query, and ii) the support of the suggested query, which measures the magnitude of answers returned in the past to this query that have attracted the attention of users. A follow-up work [4] combines various click-based, topic-based and session based ranking strategies and uses supervised learning in order to maximize the semantic similarity between the query and the recommendations. Cao et.al [5] addressed the data sparseness issue by summarizing queries into concepts by clustering a click-through bipartite. Then, from session data a concept sequence suffix tree is constructed as the query suggestion model.

Recently, deep learning techniques [16, 21] have been applied on query processing and machine translation tasks. For example, authors in [11] generate distributed language models for queries to improve the relevance in sponsored search. Authors in [21] applied recurrent neural networks on machine translation tasks and achieved state-of-the-art performance compared to traditional SMT systems. A hierarchical recurrent encoder-decoder method is proposed in [20] for the task of query auto-completion.

In this work, our approach is formulated as a learning to rewrite framework and its key component is candidate

ranking. Therefore the candidate ranking phase is similar to the learning to rank framework in terms of many aspects such as loss functions [25, 22] and ranking features [15].

3 Learning to Rewrite Framework

The query rewriting problem aims to find the query rewrites of a given query for the purpose of improving the relevance of the information retrieval system. The proposed framework formulates the query rewriting problem as an optimization problem of finding a scoring function $F(q, r)$ which assigns a score for any pair of query q and its rewrite candidate r. The framework assumes that $\mathcal{G} = \{g_1, g_2, \ldots, g_M\}$ is a set of M candidate generators. Candidate generators could be any existing query rewriting techniques and we will introduce more details in Subsection 3.1. In the candidate generating phase, we use candidate generators in $\mathcal{G}$ to generate a set of rewrite candidates for a given query q as $\mathcal{R} = \{r_1, \cdots, r_n\}$ where n is the number of generated rewrite candidates. Each pair of query q and its rewrite candidate r_i, i.e., (q, r_i), is scored by the function $F(q, r_i)$. The rewrite candidates from $\mathcal{R}$ are then ranked based on the scores $\{F(q, r_1), F(q, r_2), \ldots, F(q, r_n)\}$ in the candidate ranking phase. A key step of the learning to rewrite problem is how to obtain the scoring function F.

Let $\mathcal{F} = \{f : (q, r) \mapsto f(q, r) \in \mathcal{R}\}$ be the functional space of the scoring functions for any pair of query and rewrite candidate and $\mathcal{Q} = \{q_1, \cdots, q_m\}$ be a set of m queries. We use $\mathcal{R}_i = \{r_{i,1}, \cdots, r_{i,n_i}\}$ to denote the set of rewrite candidates of query q_i generated by $\mathcal{G}$ where n_i is the number of rewrite candidates for q_i. For each query q_i, we further assume that $\mathcal{I}_i$ is the learning target that encodes the observed information about the quality of rewrite candidates in $\mathcal{R}_i$. Note that the forms of I_i are problem-dependent that could be the label for each pair (q, r_i) or the preferences among $\mathcal{R}_i$ for q_i. With aforementioned notations and definitions, the problem of searching in $\mathcal{F}$ for a scoring function $F(q, r)$ is formally stated as the following minimization problem:

$$F = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{m} L(f, q_i, \mathcal{R}_i, \mathcal{I}_i). \quad (1)$$

The exact forms of the loss function $L(f, q_i, \mathcal{R}_i, \mathcal{I}_i)$ depends on the learning target $\mathcal{I}_i$ and we will further discuss these in Subsection 3.2. As illustrated in Figure 1, the proposed framework consists of two key steps – candidate generating and candidate ranking. In the following subsections, we elaborate them with details.

3.1 Candidate Generating

We have two ways to obtain candidate generators for candidate generating. One is to treat existing query rewriters as candidate generators. The other is to explore advanced techniques to build new candidate generators. Two candidate generators used in the paper include one query rewriter based on statistical machine translation (SMT) in [18] as well as one new generator based on deep learning techniques. More technical details about SMT based query rewriter can be found in [18] and we will also discuss how to train the rewriter in the experiment section. Next we focus on the candidate generator based on deep learning techniques.

Recently deep learning techniques have powered a number of applications from various domains such as computer vision, speech recognition and natural language processing. Meanwhile the majority of existing query rewriters are built with traditional techniques. Therefore a candidate generator based on deep learning techniques could be complementary to traditional ones and provides potentially better candidates. Recurrent Neural Network (RNN) is neural sequence model that achieves state of the art performance on many important sequential learning tasks. The long short-term memory (LSTM) is a one of the most popular RNN instance. It can learn long range temporal dependencies and mitigate the vanishing gradient problem. We propose to use the Sequence-to-Sequence LSTM model [21] to build a new candidate generator. In model training, we treat the original query as input sequence, and use its rewrite queries as target output sequences. In prediction, the most probable output sequences are obtained by a beam-search method elaborated at the end of this section and are used as the query rewrite candidates.

Given an input sequence $x_1^J = x_1, \cdots, x_J$, a standard RNN computes the hidden vector sequence $h_1^J = h_1, \cdots, h_J$ and output vector sequence $y_1^J = y_1, \cdots, y_J$ by iterating the following equations from $j = 1$ to J:

$$h_j = \mathcal{H}(W_{xh}x_j + W_{hh}h_{j-1} + b_h)$$
$$y_j = W_{hy}h_j + b_y$$

where the $W_{\cdot,\cdot}$ terms denote weight matrices, the $b_{\cdot}$ terms denote bias vectors and $\mathcal{H}$ is the hidden layer function.

For the version of LSTM used in the sequence to sequence model, the gates and cells are implemented by the following composite functions, where we followed [10]:

$$i_j = \sigma(W_{xi}x_j + W_{hi}h_{j-1} + b_i)$$
$$f_j = \sigma(W_{xf}x_j + W_{hf}h_{j-1} + W_{cf}c_{j-1} + b_f$$
$$c_j = f_j c_{j-1} + i_j tanh(W_{xc}x_j + W_{hc}h_{j-1} + b_c)$$
$$o_j = \sigma(W_{xo}x_j + W_{ho}h_{j-1} + W_{co}c_j + b_o)$$
$$h_j = o_j tanh(c_j).$$

In a sequence to sequence LSTM, we want to estimate the conditional probability $p(y_1, \cdots, y_I | x_1, \cdots, x_J)$ where $x_1, \cdots, x_J$ is an input sequence and $y_1, \cdots, y_I$ is its corresponding output sequence whose length I may differ from J. The LSTM computes this conditional probability by first obtaining the fixed dimensional representation v of the input sequence $x_1, \cdots, x_J$ given by the last hidden state of the LSTM, and then computing the probability of $y_1, \cdots, y_I$ with a standard LSTM-LM formulation whose initial hidden state is set as the representation v of $x_1, \cdots, x_J$:

$$p(y_1, \cdots, y_I | x_1, \cdots, x_J) = \prod_{i=1}^{I} p(y_i | v, y_1, \cdots, y_{i-1}), \quad (2)$$

where $p(y_i | v, y_1, \cdots, y_{i-1})$ is represented with a softmax over all the words in the vocabulary. Note that we require that each query ends with a special end-of-query symbol "<EOQ>", which enables the model to define a distribution over sequences of all possible lengths. The overall scheme is outlined in figure 2, where the shown LSTM computes the representation of the terms in the query qt_1, $qt_2, \cdots, qt_m$, <EOQ> and then uses this representation to compute the probability of rt_1, $rt_2, \cdots, rt_n$, <EOQ>.

We learn a large deep LSTM on large-scale query- rewrite query pairs. More details about how to prepare these pairs will be discussed in the experimental section. We trained it by maximizing the log probability of a correct rewrite

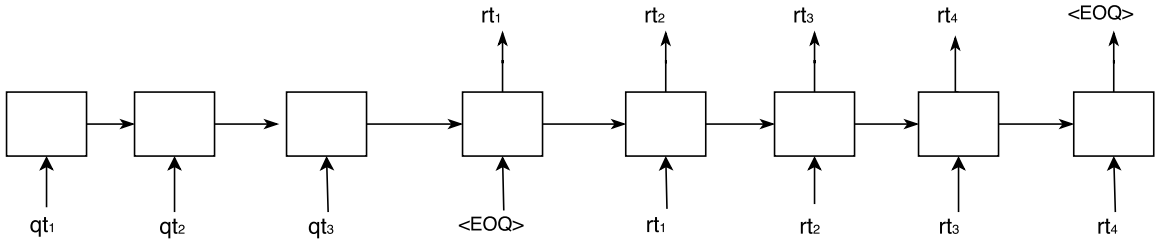

Figure 2: Scheme of Sequence to Sequence LSTM for Generating Query Rewrite

query $r = rt_1, rt_2, \cdots, rt_n, < EOQ >$ given the query $q = qt_1, qt_2, \cdots, qt_m, < EOQ >$, thus the training objective is

$$\frac{1}{|D|} \sum (q, r) \in D \log p(r|q),$$

where D is the training data set and $p(r|q)$ is calculated according to Eq. (2). Once training is complete, we feed original queries to the model and produce rewrite candidates by finding the most likely rewrites according to the LSTM – We search for the most likely query rewrites using a simple left-to-right beam search decoder instead of an exact decoder. It maintains a small number B of partial rewrites, where partial rewrites are prefixes of some query rewrite candidates. At each time-step, we extend each partial rewrite in the beam with every possible word in the vocabulary. We discard all but the B most likely rewrites according to the model's log probability. As soon as the "<EOQ>" symbol is appended to a partial rewrite, it is removed from the beam and is added to the set of complete rewrite candidates.

3.2 Candidate Ranking

A key challenge of the candidate ranking phase is to learn the scoring function that is used to rank rewrite candidates from the candidate generating phase and next we detail our solution.

3.2.1 Loss Functions

The loss function in the optimization problem for F in Eq. (1) is different with different types of the learning target $\mathcal{I}_i$. Next we define three types of loss functions according to $\mathcal{I}_i$ – point-wise, pair-wise and list-wise loss.

Point-wise loss Assuming that for each pair $(q_i, r_{i,j})$, a grade label $y_{i,j}$ is available which indicates the quality of $r_{i,j}$ as a query rewrite of q_i, the loss function that is appropriate for point-wise learning target $\mathcal{I}_i$ is:

$$L(f, q_i, \mathcal{R}_i, \mathcal{I}_i) = \sum_{j=1}^{n_i} l(f(q_i, r_{i,j}), y_{i,j}),$$

where l can be a squared error loss function, i.e., $l(x, y) = \|x - y\|^2$ if y takes numerical values, or a logistic loss function, i.e., $l(x, y) = log(1 + e^{-xy})$ if $y \in \{-1, +1\}$.

Pair-wise loss When the preference between a pair of rewrite queries $r_{i,j}$ and $r_{i,k}$ is available, the loss function for the pair-wise learning target can be constructed as the sum over pair-wise loss function. For example, let $r_{i,j} \succ r_{i,k}$ denote the learning target that $r_{i,j}$ is a preferred query rewrite to $r_{i,k}$ given the original query q_i. The pair-wise loss function has the following form:

$$L(f, q_i, \mathcal{R}_i, \mathcal{I}_i) = \sum_{r_{i,j} \succ r_{i,k}} l(f(q_i, r_{i,j}) - f(q_i, r_{i,k})),$$

where a typically used l includes the squared hinge loss function $l(t) = max(0, 1 - t)^2$ and the logistic function $l(t) = log(1 + e^{-t})$.

List-wise loss If the learning target $\mathcal{I}_i$ includes a complete order of the rewrite candidates $\mathcal{R}_i$ for q_i in terms of rewriting quality, the loss function can then be formulated to calculate the cost of the difference between the ranking order generated by the scores $f(q_i, r_{i,1}), \cdots, f(q_i, r_{i,n_i})$ and the ground-truth ranking order. For a more concrete example, let $\mathcal{I}_i$ be given in the form of $(y_i(1), y_i(2), \cdot, y_i(n_i))$ where $y_i(j)$ is the index of the candidate in the set $\mathcal{R}_i$ which is ranked at the position j. Let $\phi(q, r)$ be a mapping which retains the order in a permutation, i.e., $\phi(q_i, r_{i,y_i(1)}) > \phi(q_i, r_{i,y_i(1)}) > \cdots > \phi(q_i, r_{i,y_i(n_i)})$. The cosine loss function proposed in [17] is defined as:

$$L(f, q_i, \mathcal{R}_i, \mathcal{I}_i) = \frac{1}{2}\left(1 - \frac{\sum_{j=1}^{n_i} f(q_i, r_{i,j})\phi(q_i, r_{i,j})}{\sqrt{\sum_{j=1}^{n_i} f^2(q_i, r_{i,j})}\sqrt{\sum_{j=1}^{n_i} \phi^2(q_i, r_{i,j})}}\right).$$

More examples of the list-wise loss functions can be found in [22].

3.2.2 Generating Learning Targets

Generating the learning target $\mathcal{I}_i$ is challenging especially for a large set of queries and their corresponding rewrite candidates. One straightforward way is to use human labeling. However, it is not practical, if not impossible, to achieve this for a web-scale query rewriting application. First, it is very time and effort consuming to label millions of query rewriting pairs. Second, the relevance performance depends on many components of a search engine such as relevance ranking algorithms, thus it is extremely difficult for human editors to compare the quality of rewrite candidates. Third, for a commercial search engine, its components are typically evolved rapidly and in order to adapt to these changes, human labels are consistently and continuously needed. Therefore it is desirable for an automatic approach to generate learning target.

In this work, we specifically focus on boosting the relevance performance via query rewriting, thus the learning target should indicate the quality of the rewrite candidates from the perspective of search relevance. Intuitively a better rewrite candidate could attract more clicks to its retrieved

documents. In other words, the number of clicks on the returned document from a rewrite candidate could be a good indicator about its quality in terms of relevance. These intuitions pave us a way to develop an automatic approach to generate learning target based on the query-document click graph that is extracted from search logs.

A click graph is a query-document bipartite graph which consists of the triplets $(q, u, n_{q,u})$, where q denotes a query, u denotes a document and the edges in the bipartite graph indicate the co-click between queries and documents and the weights are co-click numbers $n_{q,u}$, i.e., the number of clicks (accumulated through a time period and a population of users) on u when the issued query is q. The click graph is constructed from user search logs, thus it contains rich information of users' understandings and behaviors about queries and documents. There are many successful applications based on the query-document graph such as query suggestion [5] and query clustering [2]. Next we introduce our approach to generate learning targets from the query-document graph.

For each query and query rewrite pair (q, r), we estimate the click numbers if we alter q to r from the query-document click graph as illustrated in Figure 3:

- We use the query r to retrieve top k documents from the search engine as $\mathcal{U} = \{u_1, u_2, \cdots, u_k\}$ where u_i indicates the ranked i-th document in the returned list. For example, we retrieve $k = 4$ documents for the rewrite candidate r_1.

- The click number of the document u_i in $\mathcal{U}$ with the information intent of q c_{r,u_i} is estimated as n_{q,u_i} if there are clicks between query q and document u_i in the query-document graph (e.g., $c_{r_1,d_2} = n_{q,d_2}$ in Figure 3) and 0 (e.g., $c_{r_2,d_2} = 0$ in Figure 3) otherwise. The rationale of using (q, u_i) instead of (r, u_i) is that r could drift the intent of q and we want to maintain the original intent. For example, if r changes the intent of q, u_i could be completely irrelevant to q and it is reasonable to estimate $c_{r,u_i} = 0$. Let $\mathcal{C}_r = \{c_{r,u_1}, c_{r,u_2}, \ldots, c_{r,u_k}\}$ be the estimated click numbers for its top k retrieved documents.

With the estimated click numbers $\mathcal{C}_r$ and document positions as shown in Figure 3, we can generate the learning target. Next we illustrate how to generate the point-wise learning target $y_{q,r}$ from $\mathcal{C}_r$. Our basic idea is to aggregate the estimated click numbers in $\mathcal{C}_r$ to a unified target $y_{q,r}$. In this paper, we investigate the following aggregating strategies:

- Clicknum: Intuitively the total click numbers can indicate the quality of a rewrite candidate in terms of relevance. Therefore the Clicknum strategy is to sum click numbers in $\mathcal{C}_r$ as $y_{q,r}$:

$$y_{q,r} := \sum_{i=1}^{k} c_{r,u_i}$$

- Discounted clicknum: In addition to click numbers, the positions of documents in the ranking list are also important. Ideally we should rank documents with a large number of clicks higher; hence we need to penalize these documents with a large number of clicks but lower positions. With these intuitions, the discounted clicknum strategy defines $y_{q,r}$ from $\mathcal{C}_r$ as:

$$y_{q,r} := \sum_{i=1}^{k} \frac{c_{r,u_i}}{i+1},$$

where the contribution of c_{r,u_i} in $y_{q,r}$ is penalized by its position in the ranking list.

- Discounted log clicknum: Click numbers in $\mathcal{C}_r$ could vary dramatically. Some have an extremely large number of clicks; while others have a few. In this scenario, the documents with large numbers of clicks could dominate the learning target. Therefore the discounted log clicknum strategy applies a log function to click numbers as:

$$y_{q,r} := \sum_{i=1}^{k} \frac{log_2(c_{r,u_i})}{i+1}.$$

- Logdiscounted log clicknum: The Discounted clicknum strategy could over-penalize the click numbers. Similar to Discounted log clicknum, Logdiscounted log clicknum also applies a log function to positions as:

$$y_{q,r} := \sum_{i=1}^{k} \frac{log_2(c_{r,u_i})}{log(i+1)}.$$

In some cases, we might not need to rewrite the original query q. For example, the quality of q could be better than that of all available rewrite candidates. Therefore in practice, we also assign the original query q as one rewrite candidate. Following the same procedure to generate the point-wise learning target for the query rewrite r, we can generate one for the original query as $y_{q,q}$. If r is better than q, the documents retrieved by r should be more relevant and the number $y_{q,r}$ should be higher than $y_{q,q}$. Similarly, if r is worse than the original query or even changes the query intent, the documents retrieved by r should be less relevant and the number $y_{q,r}$ should be lower than $y_{q,q}$. Therefore by treating the original query q as a rewrite candidate, the proposed framework allows rewriting as the original query.

We maintain the intent of the original query q by using (q, u_i) instead of (r, u_i); thus we can directly obtain pair-wise and list-wise learning targets from the point-wise targets. For example, the pair-wise labels of two rewrite candidates r_i and r_j can be obtained by comparing y_{q,r_i} and y_{q,r_j}; and similarly we can get an order of the quality of all rewrite candidates of q.

3.2.3 Feature Functions

To find the best scoring function, we need to solve the optimization problem (1) in the functional space $\mathcal{F}$, which is usually very challenging. A practical way to reduce the complexity of the optimization problem is to assume that the scoring functions in $\mathcal{F}$ have some special structures and then search for the optimal F in the constrained space. A common approach is to assume that every $f \in \mathcal{F}$ is a linear combination of m feature functions, namely:

$$f(q, r) = \sum_{i=1}^{m} \lambda_i h_i(q, r), \tag{3}$$

where each $h_i(q, r)$ is the i-th feature function and λ_i controls the contribution of $h_i(q, r)$. There are also other ap-

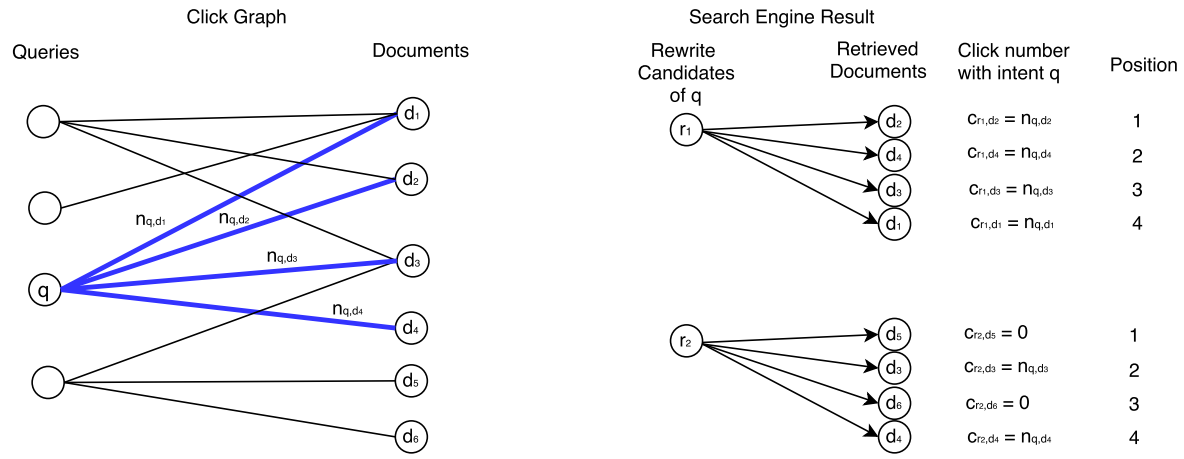

Figure 3: Generating click numbers $\mathcal{C}_r$ from the query-document click graph

proaches to build the scoring functions. For example we can assume the scoring functions in $\mathcal{F}$ are ensembles of decision trees such that the search space is constrained by the number of trees and the depth of the decision trees; and a gradient boosting algorithm can be applied to find the optimal scoring function. In this paper, we focus on the approach in Eq. (3) while we would like to leave the investigation of other approaches as one future work.

For each pair of query and rewrite candidate (q, r), we can build three groups of feature functions – (1) query features: features are extracted from only the original query q; (2) rewrite features: features are extracted from only the rewrite candidate r and (3) pair features: features are extracted from both q and r. Before introducing the details about these features, we first introduce notations we use in their definitions. Let f_q and f_r be the query frequencies obtained from the search log. Let $\mathcal{W} = \{W_1, W_2, \ldots, W_N\}$ be the dictionary with N words. We use $\mathbf{q} = \{w_{q,1}, w_{q,2}, \ldots, w_{q,N}\}$ to indicate the vector representation of q where $w_{q,i}$ is the frequency of W_i in q. Similarly we represent r as $\mathbf{r} = \{w_{r,1}, w_{r,2}, \ldots, w_{r,N}\}$. We further assume that $\mathcal{U}_q$ and $\mathcal{U}_r$ are the sets of URLs connecting to q and r in the query-document graph, respectively. The definitions and descriptions of these features are summarized in Table 1.

4 Experiment

In this section, we conduct experiments to validate the effectiveness of the proposed framework via a commercial search engine. In particular, this section mainly aims to answer the following three questions – (1) what is the quality of the rewriting query created by the proposed framework? (2) what is the impact of different learning targets on the performance of the proposed framework and (3) what are the effects of the feature functions? To answer the first question, we compare the proposed framework with the state-of-the-art baseline. Then we study the effects of different learning target generating strategies and feature functions on the performance of the proposed framework to answer the second and third questions, respectively. We begin by introducing the experimental settings.

4.1 Experimental Settings

In the candidate generating phase, we choose two candidate generators – (1) the statistical machine translation (SMT) based query rewriter [18, 8] and (2) the LSTM based query rewriter that is introduced in Subsection 3.1. We need a large parallel training data with queries and the corresponding written queries to train both SMT and LSTM. Following the common practice in [18, 8], we prepare the training data for candidate generators from the query-document click graph that is collected from the query log of a commercial search engine. We first get query and document pairs that have co-clicks from the query-document click graph. Typically titles of documents are very informative and representative; hence we can consider the title of a document as an ideal query to retrieve this document. Therefore from the extracted query and document pairs, we form a parallel training data with queries and titles. In this work, we extract 800 millions of query and title pairs from the query-document graph to train both SMT and LSTM models. To obtain the score function, we use each of these two candidate generators to generate 10 rewrite candidates for a given query and the learning targets are generated from the query-url graph with more than 2 billions of queries. Note that in this work, we only focus on the point-wise label since pair-wise and list-wise labels can be obtained from the point-wise label as discussed in Subsection 3.2.

In this work, we aim to use query rewriting to boost the relevance performance. Meanwhile it is very challenging for human editors to judge the quality of rewriting candidates in terms of relevance performance. For example, it is difficult for editors to reach an agreement about the quality of "how much tesla" and "price tesla" from the relevance perspective since the relevance performance strongly depends on many components of a given search engine. Hence we propose to evaluate the quality of query rewriting via relevance performance. We further use Discounted cumulative gain (DCG) as the metric to assess the search relevance performance. DCG has been widely used to assess relevance in the context of search engines [13]. For a ranked list of N

Table 1: Summary of Group of Features for a Pair of Query and query rewrite (q, r)

Feature Group	Feature
Query Features	h_1 – Number of words in q as $\sum_{i=1}^{N} w_{q,i}$ h_2 – Number of stop words in q: S_q h_3 – Language model score of the query q: L_q h_4 – The query frequencies of the query q: f_q h_5 – The average length of words in q: A_q
Rewrite Features	h_6 – Number of words in r as $\sum_{i=1}^{N} w_{r,i}$ h_7 – Number of stop words in r: S_r h_8 – Language model score of the query rewrite r:L_r h_9 – The query frequencies of the query rewrite r:f_r h_{10} – The average length of words in r: A_r
Pair Features	h_{11} – Jaccard similarity of URLs as $\frac{\lvert\mathcal{U}_q \cap \mathcal{U}_r\rvert}{\lvert\mathcal{U}_q \cup \mathcal{U}_r\rvert}$ h_{12} – Difference between the frequencies of the original query q and the rewrite candidate q: $f_q - f_r$ h_{13} – Word-level cosine similarity between q and r: $\frac{\sum_{i=1}^{N} w_{q,i} w_{r,i}}{\sqrt{\sum_{i=1}^{N} w_{q,i}^2}\sqrt{\sum_{i=1}^{N} w_{r,i}^2}}$ h_{14} – Difference between the number of words between q and r: $\sum_{i=1}^{N} w_{q,i} - \sum_{i=1}^{N} w_{r,i}$ h_{15} – Number of common words in q and r h_{16} – Difference of language model scores between q and r: $L_q - L_r$ h_{17} – Difference between the number of stop words between q and r: $S_q - S_r$ h_{18} – Difference between the average length of words in q and r: $A_q - A_r$

documents, we use the following variation of DCG,

$$\text{DCG}_N = \sum_{i=1}^{N} \frac{G_i}{\log_2(i+1)}$$

where G_i represents the weight assigned to the label of the document at position i, e.g. 10 for Perfect match, 7 for Excellent match, 3 for Good match, etc. Higher degree of relevance corresponds to higher value of the weight. We use the symbol *DCG* to indicate the average of this value over a set of testing queries in our experiments. To evaluate the search relevance performance, 2407 queries are sampled from the first 75% traffic of the total query list (sorted by search frequency) of a commercial search engine and segment queries into 959 top queries ($0-40\%$), 751 torso queries ($40\% - 60\%$) and 697 tail queries ($60\% - 75\%$). Web documents are retrieved from a multiple billion index and ranked by a fine-tuned commercial ranking system. For each pair of original query q and retrieved document d, highly trained human editors are requested to assign one of the five grades to determine the relevance of the document given the query. In this paper, we report the relevance performance in terms of DCG_1, DCG_3 and DCG_5.

4.2 Quality of Query Rewriting

We can reuse most of existing query rewriters in the candidate generating phase and we choose the state-of-the-art query rewriter, i.e., the SMT method proposed in [18] as one of our candidate generators, hence we compare the proposed framework with the SMT method in terms of relevance performance. Note that we do not compare the proposed framework with other query rewriters such as those in [6, 14] since we only used the SMT method as the candidate generator except the newly proposed LSTM candidate generator. For the proposed framework, we create two variants – (1) LQRW-SMT that only uses SMT to generate 20 rewrite candidates; and (2) LQRW-SMT-LSTM that uses each of SMT and LSTM to generate 10 rewrite candidates, respectively. In this experiment, we adopt the *Logdiscounted log clicknum* strategy to generate the learning target and use all feature functions defined in Table 1. More details about the effects of the learning target generating strategies and feature functions on the proposed framework will be investigated in the following subsections.

The performance comparison over all 2407 queries is reported in Table 2. We cannot report absolute DCG numbers for confidentiality reasons; hence we report the DCG relative improvement compared to SMT. Note that it is evident from t-test that all improvement is significant.

From Table 2, we make the following observations:

- With SMT as the only candidate generator, LQRW-SMT obtains remarkable improvement compared to SMT. SMT query rewriter does not consider relevance; hence rewrite candidates from SMT are not optimal for the relevance performance. While LQRW-SMT explicitly incorporates the relevance performance into query rewriting;
- By combining candidates generated by SMT and LSTM, LQRW-SMT-LSTM outperforms LQRW-SMT. Although SMT and LSTM are trained with the same set of training data, candidates from LSTM are complementary to these from SMT.

Table 2: Overall Search Performance Improvement (%) in terms of DCG Compared to the Query Rewriter SMT.

Methods	DCG_1	DCG_3	DCG_5
LQRW-SMT	+3.01%	+2.65%	+2.38%
LQRW-SMT-LSTM	+3.03%	+3.14%	+2.95%

The relevance performance of modern search engines is strongly dependent on user behavior data. However, for torso and tail queries, the user behavior data is very sparse and noisy; hence the relevance performance may be not good for these queries. It is therefore interesting to show the effects of query rewriting methods on different query traffic

bands. The performance comparisons for top, torso and tail queries are shown in Table 3 and it can be observed that:

- Both LQRW-SMT and LQRW-SMT-LSTM consistently outperform SMT for all traffic bands, i.e., top, torso and tail queries;
- The improvement of LQRW-SMT and LQRW-SMT-LSTM on torso and tail queries is higher than that on top queries. This property of the proposed framework has its practical significance since improving the relevance of torso and tail queries are usually difficult.

Table 3: Search Performance Improvement (%) in terms of DCG in Different Traffic Bands Compared to the Query Rewriter SMT.

Top			
Methods	DCG_1	DCG_3	DCG_5
LQRW-SMT	+0.72%	+1.48%	+1.51%
LQRW-SMT-LSTM	+0.66%	+1.62%	+1.84%
Torso			
LQRW-SMT	+4.64%	+3.43%	+3.23%
LQRW-SMT-LSTM	+4.93%	+4.31%	+3.97%
Tail			
LQRW-SMT	+6.14%	+4.13%	+3.53%
LQRW-SMT-LSTM	+8.22%	+5.46%	+4.55%

To deeply understand above observations, we calculate the coverage of different query rewriters in different traffic bands and the results are shown in Table 4. Note that here the coverage of a query rewriter is defined as the percentage of queries whose rewrite candidates generated by the query rewriter are different from themselves. We note that (1) all query writers have higher coverage for torso and tail queries than top queries; (2) the coverage of LQRW-SMT-LSTM is consistently higher than LQRW-SMT; and (3) the coverage of both LQRW-SMT and LQRW-SMT-LSTM increases more for torso and tail queries than top queries compared to SMT.

Table 4: Coverage of Query Writing in Different Traffic Bands.

Methods	Top	Torso	Tail
SMT	10.9%	20.7%	29.5%
LQRW-SMT	14.8%	30.4%	43.2%
LQRW-SMT-LSTM	16.1%	37.5%	51.9%

4.3 The Impact of Learning Target Generating Strategies

In Subsection 3.2, an automatic approach with four strategies is proposed to generate the learning targets. In this subsection, we investigate the impact of these learning target generating strategies on the performance of the proposed framework. We choose the *Logdiscounted log clicknum* as the basic strategy and the performance comparison of other strategies against the baseline is shown in Table 5. Note that "+" and "-" indicate performance improvement and decline, respectively.

We make the following observations from Table 5:

- *Clicknum* obtained the largest performance reduction. This result suggests that in addition to click numbers, it is also necessary and important to consider document positions when generating the learning target
- *Discounted log clicknum* outperforms *Discounted clicknum* which indicates that the learning targets could be dominated by these documents with extremely large numbers of clicks.
- The performance of *Discounted log clicknum* degrades slightly compared to *Logdiscounted log clicknum*. *Discounted log clicknum* could over-penalize the click numbers by positions.

Table 5: The Relative Performance Changes Compared to *Logdiscounted log clicknum*. Note that "+" and "-" indicate performance improvement and decline, respectively.

Learning target	DCG_1	DCG_3	DCG_5
Clicknum	-2.04%	-1.79%	-1.58%
Discounted clicknum	-0.95%	-0.73%	-0.77%
Discounted log clicknum	-0.43%	-0.34%	-0.47%

4.4 The Impact of Feature Functions

The parameter λ_i controls the contribution of the feature function h_i. Therefore in this subsection, we study the impact of feature functions via model parameters λ_i. We first rank features by the absolute values of the t-statistic for model parameters and the top 10 features are shown as follows:

1. h_{11}: Jaccard similarity of URLs of q and r;
2. h_{12}: Difference between the frequencies of q and r;
3. h_{13}: Word-level cosine similarity between q and r;
4. h_9: The query frequencies of the query rewrite r;
5. h_8: Language model score of the query rewrite r;
6. h_{14}: Difference between the number of words between q and r;
7. h_{15}: Number of common words in q and r
8. h_7: Number of stop words in r
9. h_{17}: Difference between the number of stop words between q and r
10. h_3: Language model score of the query q

Among top 10 features, there are 6 pair features, 3 rewrite features and 1 query feature. This result indicates the importance of pair features. We also investigate the performance of the proposed framework with only pair features and top 10 features, respectively. The performance changes compared to the proposed framework with all features are shown in Table 6. Note that "*" in the table indicates that the changes are not significant. When we only use pair features, the performance reduces slightly; while there is no significant performance reduction when we only use top 10 features.

Table 6: The Relative Performance Changes Compared to the Proposed Framework with All Features. Note that "*" indicates the changes are not significant.

Features	DCG_1	DCG_3	DCG_5
Pair Features	-0.44%	-0.52%	-0.61%
Top 10 Features	-0.15%*	-0.13%*	-0.17%*

5 Conclusion

This paper proposes a learning to rewrite framework that is flexible to incorporate existing query rewriting algorithms such as statistical machine translation and recent advanced methods in deep learning such as sequence to sequence LSTM. The framework formulates the query rewriting problem as an optimization problem of finding a scoring function to optimize the relevance performance explicitly. One innovation of the proposed method is that the supervised information could be cheaply constructed from the click graph. Experiments conducted on a commercial search engine demonstrate that the framework is very effective to improve search relevance.

In this future, we plan to explore the framework from three perspectives. First, we plan to investigate other forms of scoring functions. Second, we plan to investigate the effect of including more candidate generating methods in our framework. Third, we will study methods for generating pair-wise and list-wise learning targets that are complementary to the point-wise learning target generated from the click graph.

6 References

[1] I. Antonellis, H. G. Molina, and C. C. Chang. Simrank++: query rewriting through link analysis of the click graph. *Proceedings of the VLDB Endowment*, 1(1):408–421, 2008.

[2] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *Current Trends in Database Technology-EDBT 2004 Workshops*, pages 588–596. Springer, 2005.

[3] R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[4] R. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 76–85. ACM, 2007.

[5] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 875–883. ACM, 2008.

[6] H. Cui, J.-R. Wen, J.-Y. Nie, and W.-Y. Ma. Probabilistic query expansion using query logs. In *Proceedings of the 11th international conference on World Wide Web*, pages 325–332. ACM, 2002.

[7] B. M. Fonseca, P. Golgher, B. Pôssas, B. Ribeiro-Neto, and N. Ziviani. Concept-based interactive query expansion. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 696–703. ACM, 2005.

[8] J. Gao, X. He, S. Xie, and A. Ali. Learning lexicon models from search logs for query expansion. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 666–676. Association for Computational Linguistics, 2012.

[9] J. Gao and J.-Y. Nie. Towards concept-based translation models using search logs for query expansion. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, page 1. ACM, 2012.

[10] A. Graves et al. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer, 2012.

[11] M. Grbovic, N. Djuric, V. Radosavljevic, F. Silvestri, and N. Bhamidipati. Context-and content-aware embeddings for query rewriting in sponsored search. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 383–392. ACM, 2015.

[12] C.-K. Huang, L.-F. Chien, and Y.-J. Oyang. Relevant term suggestion in interactive web search based on contextual information in query session logs. *Journal of the American Society for Information Science and Technology*, 54(7):638–649, 2003.

[13] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM TOIS*.

[14] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web*, pages 387–396. ACM, 2006.

[15] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[16] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[17] T. Qin, X.-D. Zhang, M.-F. Tsai, D.-S. Wang, T.-Y. Liu, and H. Li. Query-level loss functions for information retrieval. *Information Processing & Management*, 44(2):838–855, 2008.

[18] S. Riezler and Y. Liu. Query rewriting using monolingual statistical machine translation. *Computational Linguistics*, 36(3):569–582, 2010.

[19] S. Riezler, Y. Liu, and A. Vasserman. Translating queries into snippets for improved query expansion. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 737–744. Association for Computational Linguistics, 2008.

[20] A. Sordoni, Y. Bengio, H. Vahabi, C. Lioma, J. Grue Simonsen, and J.-Y. Nie. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 553–562. ACM, 2015.

[21] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances*

in neural information processing systems, pages 3104–3112, 2014.

[22] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pages 1192–1199. ACM, 2008.

[23] J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 4–11. ACM, 1996.

[24] W. V. Zhang and R. Jones. Comparing click logs and editorial labels for training query rewriting. In *WWW 2007 Workshop on Query Log Analysis: Social And Technological Challenges*, 2007.

[25] Z. Zheng, K. Chen, G. Sun, and H. Zha. A regression framework for learning ranking functions using relative relevance judgments. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 287–294. ACM, 2007.