Context-Aware Query Rewriting for Improving Users’ Search Experience on E-commerce Websites

Simiao Zuo*, Qingyu Yin†, Haoming Jiang†, Shaohui Xi†, Bing Yin†, Chao Zhang‡ and Tuo Zhao*
{simiaozuo, chaozhang, tourzhao}@gatech.edu
{qingyy, jhaoming, shaohux, alexbyin}@amazon.com
*Georgia Institute of Technology †Amazon

Abstract

E-commerce queries are often short and ambiguous. Consequently, query understanding often uses query rewriting to disambiguate user-input queries. While using e-commerce search tools, users tend to enter multiple searches, which we call context, before purchasing. These history searches contain contextual insights about users’ true shopping intents. Therefore, modeling such contextual information is critical to a better query rewriting model. However, existing query rewriting models ignore users’ history behaviors and consider only the instant search query, which is often a short string offering limited information about the true shopping intent. We propose an end-to-end context-aware query rewriting model to bridge this gap, which takes the search context into account. Specifically, our model builds a session graph using the history search queries and their contained words. We then employ a graph attention mechanism that models cross-query relations and computes contextual information of the session. The model subsequently calculates session representations by combining the contextual information with the instant search query using an aggregation network. The session representations are then decoded to generate rewritten queries. Empirically, we demonstrate the superiority of our method to state-of-the-art approaches under various metrics.

1 Introduction

Query rewriting is a task where a user inputs a potentially problematic query (e.g., typos or insufficient information), and we rewrite it to a new one that better matches the user’s real shopping intent. This task plays an important role in e-commerce query understanding, where without proper rewriting, search engines often return undesired items, rendering the search experience unsatisfactory.

One major issue that impedes query rewriting is the ambiguity of queries. For example, Figure 1 (left) demonstrates searching for “bumblebee costumes” without considering search context. From the query alone, it is implausible to tell if the user’s intent is for costumes of actual bumblebee (i.e., the animal) or the character from the movie franchise. This type of ambiguity is common in e-commerce search, where queries are usually short (only 2-3 terms) and insufficiently informative (He et al., 2016b). Therefore, it is not possible to disambiguate queries using only the instant search. A common solution is to use statistical rules to differentiate the possible choices. Specifically, in our example, suppose a total of 100 users entered the “bumblebee costumes” query, and 70 of them eventually purchased the movie character costume. When a new user searches for the same query, the recommended products will consist of 70% movie character costumes and 30% animal costumes. This procedure is problematic because each user has a specific intent, i.e., either the movie character costume or the animal costume, but rarely both, which the aforementioned method fails to address.

We propose to explore contextual information from users’ history searches to resolve the query ambiguity issue. Taking the “bumblebee costumes” example again, in Figure 1 (right), suppose a rewriting model recognizes that the user searched for “Transformers movie” earlier, then it could infer
that the user’s purchase intent is the movie character costume, and hence can remove the input ambiguity. There have been existing works that utilize search logs for query rewriting. For example, Wang and Zhai (2007, 2008) use traditional TF-IDF-based similarity metrics to capture relational information among the user’s history searches. These approaches are too restrictive to handle the increasingly complex corpus nowadays. As such, the rewritten queries significantly differ from the original one in intent. More recently, neural network-based query rewriting algorithms (He et al., 2016b; Xiao et al., 2019; Yang et al., 2019) are proposed. Most of such approaches employ a multi-stage training approach. Consequently, they involve complicated hand-crafted features or require excessive human annotations for the intermediate features (sometimes both).

To overcome the drawbacks of existing methods, we propose an end-to-end context-aware query rewriting algorithm. Our model’s backbone is the Transformer (Vaswani et al., 2017). In our context-aware model, the Transformer encoder learns representations for individual history queries. The representations are further transformed to carry cross-query relational information using a graph attention mechanism (GAT, Velickovic et al. 2018). The GAT computes contextual information of a session based on a session graph, where its nodes contain the history queries and the tokens contained in the history queries. After obtaining the contextual information from the GAT, it is aggregated with the instant search using an aggregation network. The augmented information is subsequently fed into the Transformer decoder to generate rewritten queries.

Our proposed method improves upon existing works from three aspects. First, our model does not involve recursion, unlike conventional recurrent neural network-based approaches (He et al., 2016b; Yang et al., 2019; Xiao et al., 2019). This facilitates training deep models containing dozens of layers capable of capturing high-order information. Second, our end-to-end sequence-to-sequence learning formulation eliminates the necessity of excessive labeled data. Previous approaches (Yang et al., 2019; Xiao et al., 2019) require the judgment of “semantic similarity”, and thus crave for human annotations, which are expensive to obtain. In contrast, our method uses search logs as supervision, which does not involve human effort, and are cheap to acquire. Third, our method can leverage powerful pre-trained language models, such as BART (Lewis et al., 2020). Such models contain rich semantic information and are successful in numerous natural language processing tasks (Devlin et al., 2019; Liu et al., 2019; Radford et al., 2019).

We demonstrate the effectiveness of our method on in-house data from an online shopping platform. Our context-aware query rewriting model outperforms various baselines by large margins. Notably, comparing with the best baseline method (Transformer-based model), our model achieves 11.6% improvement under the MRR (Mean Reciprocal Rank) metric and 20.1% improvement under the HIT@16 metric (a hit rate metric). We further verify the effectiveness of our approach by conducting online A/B tests.

2 Related Works

One line of work uses statistical methods. For example, Cui et al. (2002, 2003) extract probabilistic correlations between the search queries and the product descriptions. Other works extract features that are related to the user’s current search (Huang et al., 2003; Huang and Efthimiadis, 2009), or from relational information among the user’s history searches (Billerbeck et al., 2003; Baeza-Yates and Tiberi, 2007; Wang and Zhai, 2007; Cao et al., 2008; Wang and Zhai, 2008). There are also statistical machine translation-based models (Riezler et al., 2007; Riezler and Liu, 2010) that employ sequence-to-sequence approaches. The aforementioned statistical methods suffer from unreliable extracted features, such that the rewritten queries differ from the original one in intent.

Another line of work focuses on neural query rewriting models (He et al., 2016b; Xiao et al., 2019; Yang et al., 2019). These models adopt recurrent neural networks (RNNs, Hochreiter and Schmidhuber 1997; Sutskever et al. 2014) to learn a vectorized representation for the user’s search query, after which KNN-based methods are used to find queries that yield similar representations. One major limitation is that the rewritten queries are limited to the previously presented ones. Also, these methods often involve complicated and ungrounded feature function designs, e.g., He et al. (2016b) and Xiao et al. (2019) hand-crafted 18 feature functions, or require excessive labeled data (Yang et al., 2019). Other works (Sordoni et al., 2015; Dehghani et al., 2017; Jiang and Wang, 2018) use RNNs for generative query suggestion,
but they inherit the weaknesses of RNNs and yield unsatisfactory performance in practice.

Note that Grbovic et al. (2015) construct context-aware query embeddings using word2vec (Mikolov et al., 2013). In their approach, an embedding is learned for each distinct query in the dataset. As such, the quality of the learned embeddings rely heavily on the number of occurrences of each query. This method is not applicable to our case because in our dataset, almost all the queries are distinct.

3 Problem Setup

The session data are collected from search logs. First, we collect all the searches from a specific user within a time window, and we call the searches a “session”. After the user purchases a product, the session ends, i.e., we do not consider subsequent queries and behaviors after a purchase happens. This is because after a purchase, the user’s intent often change. Note that different sessions may be collected from different users.

Each session contains multiple searches from the same user. We call the last query in the session the “target” query, the second to the last query the “source” (or the “instance”) query, and the others the “history” queries. The intuition behind this is that because sessions always end with a purchase, the last search (i.e., the target) reflects the user’s real intent. When the user enters the second to the last search (i.e., the source), if we can rewrite it to the target query, the user’s intent will be fulfilled.

Below is an example of a search session. From the history queries, the user is interested in car related banners/posters. The source query contains a typo (“doger” is a baseball team) and we should rewrite it to the target query (“dodge posters”).

**History**: {dodge banners; mopar poster}

**Source (Instance)**: dodger posters

**Target**: dodge posters

We collect about 3 million (M) sessions, where each session consists of at least 3 history queries, a source query (i.e., the one we need to rewrite), and a target query (i.e., the ground-truth query that is associated with the purchase). We have roughly 18.7M queries, and on average, each session contains 4 history queries. Query rewriting is consequently formulated as a sequence-to-sequence learning problem. We highlight that per our formulation, we do not need human annotations, unlike existing approaches.

4 Method

Figure 2 illustrates our context-aware query rewriting model. The model contains four parts: a conventional Transformer (Vaswani et al., 2017) encoder, a graph attention mechanism (Velickovic et al., 2018) that captures the user’s purchase intent, an aggregation network that encodes the history searches, and a conventional Transformer decoder that generates the rewritten query candidates.

4.1 Transformer Encoder

For a given source query, we first pad it with a <boq> (begin-of-query) token. Then, we pass the padded query through a Transformer encoder, after which we have its hidden representation $H_s \in \mathbb{R}^{L_s \times d}$. Here $L_s$ is the length of the padded query, and $d$ is the hidden dimension. We also pass all the history queries corresponding to this source query through the encoder, and we have the history query representation $U_h \in \mathbb{R}^{N_h \times L_h \times d}$, where $N_h$ is the number of history queries and $L_h$ is the padded length. More details are presented in Appendix A.

4.2 Contextual Information from Session Graphs

After we obtain the history query representations $U_h$, the next step is to refine them. Such refinement is necessary because the Transformer encoder considers the history queries separately, such that their interactions are not taken into account. However, since each search depends on its previous searches in the same session, modeling cross-query relations are imperative for determining the user’s purchase
We use a multi-head graph attention mechanism (Velickovic et al., 2018; Wang et al., 2020) to capture contextual information from $U_h$.

4.2.1 Session Graph Construction

First we specify how to build a graph for each session, which we call the session graph. Suppose we have a session that contains three history queries:

$$Q_1 : \{ \text{Search query} : T_1, T_3 \},$$
$$Q_2 : \{ \text{Search query} : T_1, T_2, T_3 \},$$
$$Q_3 : \{ \text{Search query} : T_1, T_2, T_4, T_5 \},$$

where $Q_1, Q_2, Q_3$ are the three queries, and $T_1, \cdots, T_5$ are the five tokens that appear in the three queries. Recall Section 3 for the problem setup. Figure 3 (left) illustrates the session graph.

4.2.2 Node Representations

The next step is to refine the node representations. Each of the nodes in the session graph has its own representation. The token representations are simply the corresponding representations of the tokens, extracted from the token embedding matrix. The query representations are the representations of the $<\text{boq}>$ token in each padded history query, i.e., the representation of the $Q_1$ query node in Figure 3 is found by $U_h[0, 0, :] \in \mathbb{R}^d$. Denote $G_q = \{ q_t \}_{t \in \mathbb{N}_q}$ and $G_t = \{ t_i \}_{i=1}^{N_t}$ the sets of representations for the query and token nodes, respectively. Here $N_q$ is the number of query nodes and $N_t$ is the number of token nodes. Note that all the node representations have the same size, i.e., $q_t, t_i \in \mathbb{R}^d$.

4.2.3 Update Node Representations

We use a multi-head graph attention mechanism to update the node representations. For simplicity, denote $N_g = N_q + N_t$ the number of distinct nodes in the session graph, and $G = G_q \cup G_t = \{ g_i \}_{i=1}^{N_g}$ the set of all the node representations.

With the above notations, a single-head graph attention mechanism is defined as

$$h_i = g_i + \text{ELU} \left( \sum_{j \in N_i} \alpha_{ij} W_q g_j \right),$$

where $\alpha_{ij} = \frac{\exp(z_{ij})}{\sum_{\ell \in N_i} \exp(z_{ij})}$.

$$z_{ij} = \text{LeakyReLU} \left( W_{a} [W_q g_i; W_k g_j] \right).$$

Here $\text{ELU}(x) = x \cdot 1 \{ x > 0 \} + (\exp(x) - 1) \cdot 1 \{ x \leq 0 \}$ is the exponential linear unit, $N_i$ denotes the neighbor of the $i$-th node, and $W_a, W_q, W_k, W_v$ are trainable weights. Note that a residual connection (He et al., 2016a) is added to the last equation in Eq. 2. This has proven to be an effective technique to prevent gradient vanishing, and hence, to stabilize training.

The session graph only induces attention between nodes that are connected. For example, in Figure 3 (right), the model updates $Q_1$ and $Q_2$ using $T_3$, while $Q_3$ is unchanged, i.e., $N_{T_3} = \{ Q_1, Q_2 \}$. A multi-head graph attention mechanism is then defined as the concatenation of $[h_1^1, h_1^2, \cdots, h_1^K]$, where $K$ is the number of heads, and each of the $h_i$ is calculated via Eq. 2.

The token node representations and the query node representations are updated iteratively. First, we update the token representations ($G_t$) using the query representations ($G_q$), in order that the tokens acknowledge to which queries they belong. Then, $G_q$ is re-computed using the updated version of $G_t$, which essentially evaluates cross-query relations, using the token nodes as intermediaries. Note that the graph attention mechanism (GAT) used in each of the two steps are distinct, i.e., there are two different sets of weights $[W_a, W_q, W_k, W_v]$.

Eventually, we obtain the updated vectorized representations $\{ h_i \}_{i=1}^{N_g}$ for all the nodes, and we treat them as the contextual information of the session.

We remark that the GAT mechanism explicitly models cross-query relations by associating query representations with word representations. Such an approach is fundamentally different from existing methods, where the relations are either ignored (e.g., conventional Transformer attention) or captured via recursion (e.g., RNN-based approaches).

4.3 Session Representations

Recall that we pass the source query through a Transformer encoder and obtain $H_s \in \mathbb{R}^{L_s \times d}$. The matrix $H_s$ contains representations for all the tokens in the source query. We use that of the
prepended \texttt{<boq>} token as the representation of the source query, which is denoted \( h_s \in \mathbb{R}^d \). We adopt an aggregation network to extract useful information with respect to \( h_s \) from the contextual information \( \{ h_i \}_{i=1}^{N_h+N_t} \). The network employs an attention mechanism that determines to what extent each vector \( h_i \) contributes to the source query \( h_s \).

Figure 4 illustrates the architecture of the aggregation network. Concretely,

\[
H_{\text{sess}} = H_s + \sum_{i=1}^{N_h} \alpha_i W_v h_i, \tag{3}
\]

where \( \alpha_i = \frac{\exp(z_i)}{\sum_{j=1}^{N_h} \exp(z_j)} \), \( z_i = (W_k h_i)\top h_s \), \( W_k \) and \( W_v \) are trainable weights. The summation in the last equation in Eq. 3 is conducted row-wise, wherein \( H_{\text{sess}}, H_s \in \mathbb{R}^{L_s \times d} \), and \( v \in \mathbb{R}^d \). The matrix \( H_{\text{sess}} \) serves as the representation of the session. Intuitively, by incorporating the aggregation network, we can filter out redundant information from the session history and only keep the ones pertinent to the source query.

After the Transformer encoder, the graph attention mechanism, and the aggregation network, we obtain \( H_{\text{sess}} \), the session representation that contains information on both the source query and its history searches. Subsequently, \( H_{\text{sess}} \) is fed into the Transformer decoder to generate rewritten query candidates. The algorithm is detailed in Algorithm 1 in Appendix D.

5 Experiments

We conduct experiments on some in-house data. We implement two methods with different model architectures: Transformer+Aggregation+Graph and BART+Aggregation+Graph. The first one is constructed in the previous section, and the second one employs a fine-tuning approach instead of training-from-scratch. The training details are deferred to Appendix B.3. More experimental results are shown in Appendix C.

5.1 Baselines

For baselines with pre-training, we use MeshBART (Chen and Lee, 2020) and BART (Lewis et al., 2020). For baselines without pre-training, we use LQRW (He et al., 2016b), HRED (Sordoni et al., 2015) and MeshTransformer (Chen and Lee, 2020) (a variant of MeshBART where we train the model from scratch). We also compare our algorithm with two model variants: Transformer+Aggregation and BART+Aggregation, where we use the aggregation network but not the GAT mechanism. Please refer to Appendix B.1 for details.

5.2 Evaluation Metrics

We use BLEU, MRR (Mean Reciprocal Rank), HIT@1, and HIT@16 to evaluate the query rewriting models. For all metrics except BLEU, we report the gains over the the results calculated by using only source queries. We remark that MRR, HIT@1, and HIT@16 (the percentage that the actual product is ranked within the first 16 products i.e., the first page when we search the rewritten query) are more important than BLEU, because MRR and HIT are directly linked to user experience. Please refer to Appendix B.2 for details about these metrics.

5.3 Experimental Results

Table 1 summarizes experimental results. Recall that in our formulation, we rewrite a source query to a target query. The “target query” entry in Table 1 is the performance gain of the ground truth target query, i.e., this entry signifies upper bounds of performance gain that any model can achieve.

We can see that the attention-based models (i.e., BART, MeshBART, Transformer and MeshTransformer) outperform the recurrent neural network-based approach (i.e., LQRW and HRED). This is because RNNs suffer from forgetting and training issues. In contrast, Transformer-based models use the attention mechanism instead of recursion to capture dependencies, which has proven to be more effective. Moreover, by aggregating history searches, BART+Aggregation and Transformer+Aggregation consistently outperform their vanilla alternatives. Essentially performance of these two methods indicate that integrating history queries into training is critical. The performance is further enhanced by incorporating the session graphs. Specifically, Transformer+Aggregation+Graph achieves the best performance under almost all the metrics. Notice that the HIT@16 metric gain improves from +15.9 to
Table 1: Experimental results. The results of MRR, HIT@1, and HIT@16 are shown as gain over the source query. The best results are shown in **bold**.

<table>
<thead>
<tr>
<th>Metric</th>
<th>#Candidates=5</th>
<th>#Candidates=10</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Query</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target Query</td>
<td>+16.1</td>
<td>+16.1</td>
<td>─</td>
</tr>
</tbody>
</table>

**Baseline methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>MRR</th>
<th>HIT@1</th>
<th>HIT@16</th>
<th>MRR</th>
<th>HIT@1</th>
<th>HIT@16</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>LQRW</td>
<td>+3.5</td>
<td>+2.5</td>
<td>+6.4</td>
<td>+6.8</td>
<td>+4.9</td>
<td>+12.6</td>
<td>29.4</td>
</tr>
<tr>
<td>HRED</td>
<td>+4.7</td>
<td>+3.2</td>
<td>+8.4</td>
<td>+8.1</td>
<td>+5.7</td>
<td>+14.2</td>
<td>25.7</td>
</tr>
<tr>
<td>BART</td>
<td>+4.6</td>
<td>+3.1</td>
<td>+8.2</td>
<td>+8.2</td>
<td>+5.5</td>
<td>+14.8</td>
<td>30.9</td>
</tr>
<tr>
<td>Transformer</td>
<td>+4.3</td>
<td>+2.6</td>
<td>+9.2</td>
<td>+8.5</td>
<td>+5.6</td>
<td>+15.9</td>
<td>25.3</td>
</tr>
<tr>
<td>MeshBART</td>
<td>+5.0</td>
<td>+3.8</td>
<td>+8.7</td>
<td>+8.3</td>
<td>+5.8</td>
<td>+14.3</td>
<td>31.7</td>
</tr>
<tr>
<td>MeshTransformer</td>
<td>+4.0</td>
<td>+2.7</td>
<td>+8.4</td>
<td>+8.3</td>
<td>+5.6</td>
<td>+15.7</td>
<td>25.9</td>
</tr>
</tbody>
</table>

**Our methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>MRR</th>
<th>HIT@1</th>
<th>HIT@16</th>
<th>MRR</th>
<th>HIT@1</th>
<th>HIT@16</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>BART+Aggregation</td>
<td>+6.3</td>
<td>+3.9</td>
<td>+10.9</td>
<td>+9.7</td>
<td>+6.4</td>
<td>+17.1</td>
<td>31.9</td>
</tr>
<tr>
<td>Transformer+Aggregation</td>
<td>+5.2</td>
<td>+2.9</td>
<td>+10.8</td>
<td>+10.2</td>
<td>+7.0</td>
<td>+17.3</td>
<td>27.2</td>
</tr>
<tr>
<td>BART+Aggregation+Graph</td>
<td><strong>+6.9</strong></td>
<td><strong>+4.6</strong></td>
<td><strong>+11.8</strong></td>
<td>+10.5</td>
<td>+7.5</td>
<td>+17.6</td>
<td>32.9</td>
</tr>
<tr>
<td>Transformer+Aggregation+Graph</td>
<td><strong>+6.6</strong></td>
<td><strong>+4.6</strong></td>
<td><strong>+12.0</strong></td>
<td><strong>+11.6</strong></td>
<td><strong>+8.3</strong></td>
<td><strong>+20.1</strong></td>
<td>28.2</td>
</tr>
</tbody>
</table>

+20.1 when employing both the aggregation network and the session graph formulation for the Transformer-based models. We highlight that the graph attention mechanism can directly captures cross-query relations, which is implausible for all the baselines. We can see that this property indeed contributes to model performance, i.e., HIT@16 increases from +17.3 to +20.1 when we equip Transformer+Aggregation with the GAT mechanism.

Notice that BLEU is not a definitive metric. For example, the MRR and HIT metrics of HRED are consistently higher than those of LQRW, even though the BLEU score of the former is significantly lower than the latter. Also, compared with Transformer-based models, the BLEU score is consistently higher when using the BART model as the backbone. This is because a pre-trained language model contains more semantic information. However, the MRR and HIT metrics of the BART-based models are slightly worse than those of the Transformer-based models.

However, the BLEU score is comparable for models with the same backbone. For example, for Transformer vs. Transformer+Aggregation vs. Transformer+Aggregation+Graph, the BLEU scores are 25.3 vs. 27.2 vs. 28.2. Such a tendency coincides with the online metrics. We observe the same results from BART-based models.

5.4 Online A/B Test

We conduct online A/B experiments on a large-scale e-commerce shopping platform with our query rewriting models. For a given search query within a session, we generate one reformulated query using the proposed model, and we feed both the original query and rewritten query into the search system. Experiments are conducted over five days, during which our system processed over 30 million sessions. The proposed method improves business metrics in terms of revenue; and also significantly decreases the number of reformulated searches. This indicates that the rewritten queries better meet customers’ shopping intent since customers are able to find their desired products with less number of searches.

5.5 Case Studies

⋄ Advantages of leveraging history information

Two examples are shown in Table 2. The first example is error correction. In the example, the customer wishes to purchase dodge (a car brand) posters, but she mistakenly searches for dodger (a baseball team) posters. Without history information, it is impossible to determine the customer’s true intent.
### Table 3: Two examples of generated queries and their associated likelihood.

<table>
<thead>
<tr>
<th>Type</th>
<th>Query</th>
<th>Likelihood</th>
<th>Query</th>
<th>Likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>History</td>
<td>iphone 11 pro case pokemon;</td>
<td>—</td>
<td>colorado 2005 tail lights;</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>iphone 11 pro case eevee;</td>
<td>—</td>
<td>colorado 2005 door</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>iphone 11 pro case hetalia;</td>
<td>—</td>
<td>colorado 2005 accessories</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>iphone 11 pro case sailor moon</td>
<td>—</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source</td>
<td>iphone 11 pro case snow leopard</td>
<td>—</td>
<td>colorado headlights</td>
<td>—</td>
</tr>
<tr>
<td>Target</td>
<td>iphone 11 pro case tiger</td>
<td>—</td>
<td>colorado 2005 headlights</td>
<td>—</td>
</tr>
<tr>
<td>Rewritten</td>
<td>iphone 11 pro case disney</td>
<td>0.497</td>
<td>2005 colorado headlights</td>
<td>0.566</td>
</tr>
<tr>
<td></td>
<td>iphone 11 pro case sailor moon</td>
<td>0.492</td>
<td>colorado headlights 2005</td>
<td>0.458</td>
</tr>
<tr>
<td></td>
<td>iphone 11 pro case harry potter</td>
<td>0.445</td>
<td>colorado headlights led</td>
<td>0.357</td>
</tr>
<tr>
<td></td>
<td>iphone 11 pro case</td>
<td>0.440</td>
<td>colorado headlights assembly</td>
<td>0.301</td>
</tr>
<tr>
<td></td>
<td>iphone 11 pro case cute</td>
<td>0.419</td>
<td>colorado tail lights</td>
<td>0.289</td>
</tr>
<tr>
<td></td>
<td>iphone 11 pro case leopard</td>
<td>0.391</td>
<td>colorado headlights housing</td>
<td>0.237</td>
</tr>
<tr>
<td></td>
<td>iphone 11 pro case clear</td>
<td>0.379</td>
<td>colorado led headlights</td>
<td>0.234</td>
</tr>
<tr>
<td></td>
<td>iphone 11 pro case disney princess</td>
<td>0.372</td>
<td>2004 colorado headlights</td>
<td>0.230</td>
</tr>
<tr>
<td></td>
<td>iphone 11 pro case pink</td>
<td>0.364</td>
<td>colorado 2004 headlights</td>
<td>0.214</td>
</tr>
<tr>
<td></td>
<td>iphone 11 pro case totoro</td>
<td>0.353</td>
<td>colorado headlights 2004</td>
<td>0.208</td>
</tr>
</tbody>
</table>

However, by looking at session histories, we find that all the previous searches are related to automobiles (e.g., dodge and mopar), and therefore the query should be rewritten to “dodge posters”. Our model successfully captures this pattern. Notice that the rewritten query without leveraging context does not match the user’s intent.

The second example is keyword refinement. In the example, by looking at the history searches, it is obvious that the customer wishes to find phone cases, instead of phones. However, this intent is impossible to capture by using only the source query. Our model automatically adds the keyword “case” to the source query and matches the target query. On the other hand, without the context information, the rewritten result is not satisfactory.

◊ **Diversity of query generation** Table 3 demonstrates two examples. In the first example (the left three columns), notice that our model can grep information from history queries, e.g., “iphone 11 case sailor moon”, and can delete keywords that are deemed insignificant or too restrictive, e.g., “iphone 11 case leopard” instead of “snow leopard”. Also, our model can effectively capture domain information. For example, some of the history query keywords (e.g., pokemon, eevee) are often described as “cute”, and our model recommends this keyword. All the history keywords are from Japanese anime series, therefore our model suggests another popular character, “totoro”. Additionally, the “disney” and “disney princess” keywords are generated based on the interest to virtual characters. Finally, notice that the likelihood of all the suggested queries is similar, which means our model cannot single out a significantly better query than the others. Therefore our model generated a diverse group of queries.

In the second example (the right two columns), the generated query successfully matches the target query. Note that the top two generated queries have high likelihood, and the likelihood decreases drastically as the suggested queries become more and more implausible. In this example, the first query is 172% more likely than the tenth query, whereas this number is only 41% in the previous example. This suggests that our model can differentiate between good quality suggestions and poor quality alternatives.

## 6 Conclusion and Discussion

We propose an end-to-end context-aware query rewriting model that can efficiently leverage user’s history behavior. Our model infers a user’s purchase intent by modeling her history searches as a graph, on which a graph attention mechanism is applied to generate informative session representations. The representations are subsequently decoded into rewritten queries. Our proposed session graph can be extended to incorporate more information. Here, we present a bipartite graph, which contains words and queries. Additional components can be added as extra layers. For example, we can add product information such as categories to the session graph, which will create 3-partite session graphs (word, query and product).
References


Huanhuan Cao, Daxin Jiang, Jian Pei, Qi He, Zhen Liao, Bodo Billerbeck, Falk Scholer, Hugh E Williams, and Ricardo Baeza-Yates and Alessandro Tiberi. 2007. Ex-

References


A Transformer Encoder Details

For a given source query, we first pad it with a $<boq>$ (begin-of-query) token. Then, we pass the padded query through a token embedding layer and a position embedding layer, and we obtain $Y_s \in \mathbb{R}^{L_s \times d}$. Here $L_s$ is the length of the padded source query, and $d$ is the embedding dimension. Note that the position embedding can either be a sinusoidal function or a learned matrix.

After the initial embedding layers, we pass $Y_s$ through the self-attention module. Specifically, we compute attention output $S$ by

$$S = \text{Softmax} \left( \frac{QK^T}{\sqrt{d_k}} V \right),$$

where $Q = Y_s W_q$, $K = Y_s W_k$, $V = Y_s W_v$. \hspace{1cm} (4)

Here $W_q, W_k \in \mathbb{R}^{d \times d_K}$, $W_v \in \mathbb{R}^{d \times d_V}$ are learnable weights. In practice we use multi-head self-attention to increase model flexibility. To facilitate this, different attention outputs $S_1, \ldots, S_H$ are computed using different sets of weights $\{W^h_q, W^h_k, W^h_v\}_{h=1}^H$. The final attention output is

$$S = [S_1, S_2, \ldots, S_H] W_o,$$ \hspace{1cm} (5)

where $W_o \in \mathbb{R}^{Hd_v \times d}$ is a learnable aggregation matrix. The attention output is then fed through a position-wise feed-forward neural network to generate encoded representation $H_s \in \mathbb{R}^{L_s \times d}$ for the source query:

$$H_s = \text{ReLU} \left( S W_{\text{FFN}}^1 + b^1 \right) W_{\text{FFN}}^2 + b^2.$$ \hspace{1cm} (6)

Here $\{W_{\text{FFN}}^1, W_{\text{FFN}}^2, b^1, b^2\}$ are weights of the neural network. Equations 4, 5, and 6 constitute as an encoder block. In practice we stack multiple encoder blocks to build the Transformer encoder, as demonstrated in Figure 2.

For the history queries in this session, we also pad them with $<boq>$ tokens. Suppose that we have $N_h$ padded history queries (recall a session contains multiple history queries), and their respective length is denoted by $L^1_h, \ldots, L^{N_h}_h$. We pad the history queries to the same length, and we obtain the history query matrix $X_h \in \mathbb{R}^{N_h \times L_h}$, where $L_h = \max \{L^1_h, \ldots, L^{N_h}_h\}$. Then, following the same procedures as encoding the source query, we pass $X_h$ through the embedding layers and the encoder blocks, after which we obtain the history query representations $U_h \in \mathbb{R}^{N_h \times L_h \times d}$.

B Experiments Details

B.1 Baselines

The baselines are split into two groups: without pre-training and with pre-training. For the w/o pre-training group, we build the following models:

- **Hierarchical Recurrent Encoder-Decoder (HRED)** (Sordoni et al., 2015) employs a hierarchical recurrent neural network for generative query suggestion. The model is a step forward from its predecessors in that HERD is sensitive to the order of queries and the method is able to suggest rare and long-tail queries.

- **Learning to Rewrite Queries (LQRW)** (He et al., 2016b) is one of the first methods that applies deep learning techniques to query rewriting. Specifically, the LQRW model combines a sequence-to-sequence LSTM (Hochreiter and Schmidhuber, 1997; Sutskever et al., 2014) model with statistical machine translation (Riezler and Liu, 2010) techniques to generate queries. The candidates are subsequently ranked using hand-crafted feature functions.

- **Transformer** (Vaswani et al., 2017) has achieved superior performance in various sequence-to-sequence (seq2seq) learning tasks. To adopt Transformer to query rewriting, we treat the source query as the source-side input, and the target query as the target-side input. Then we train a model using only these constructed inputs, similar to machine translation. Note that this setting resembles most of the existing works. We adopt the Transformer-base architecture, which contains about 72M parameters.

- **MeshTransformer** (Chen and Lee, 2020) is a variant of MeshBART, where the pre-trained BART module is replaced by a Transformer and the model is trained from scratch. The method concatenates history queries to the source query in order to integrate contextual information. See the MeshBART method below for details.

- **Transformer+Aggregation** is the model where we use the aggregation network to encode history search queries, i.e., without the graph attention mechanism. Specifically, we first obtain the representations of the source query and the history queries from the Transformer encoder. Then, we extract information related to the source query from the history representations using an aggregation network. Such information is added to the source representation, and we follow a standard decoding
procedure using these two factors. See Section 4.3 for details.

The second group of methods adopt pre-trained language models for query rewriting.

- **BART** (Lewis et al., 2020) is a pre-trained seq2seq model. We adopt this particular model instead of, for example, BERT (Devlin et al., 2019) or GPT-2 (Radford et al., 2019), because we treat query rewriting as a seq2seq task. And the aforementioned architectures have either the Transformer encoder (e.g., BERT) or the Transformer decoder (e.g., GPT-2), but not both. In our experiments, BART is fine-tuned in a setting similar to training the Transformer model. We adopt the BART-base architecture in all the experiments, which contains about 140M parameters.

- **MeshBART** (Chen and Lee, 2020) is a BART-based model that first concatenates the history queries to the source query, and then feeds the concatenated input to a pre-trained BART model for query generation. Note that the original method requires click information. We remove this component as the proposed method do not need such data.

- **BART+Aggregation** is similar to Transformer+Aggregation, except we replace the Transformer backbone with the pre-trained seq2seq BART model.

### B.2 Evaluation Metrics

We use the BLEU score (Post, 2018) as an evaluation metric. This metric is constantly used to evaluate the quality of translation. We adopt it here because similar to machine translation, we formulate query rewriting as a seq2seq learning task. The correlation between the rewritten query and the target query reflects the model’s ability to capture the user’s purchase intent.

The MRR metric describes the accuracy of the rewritten queries. For each source query in the test set, we generate 10 candidate queries \( r_1, \ldots, r_{10} \). Then we search each of these candidates using our production search engine, and we obtain the returned products, of which we only keep the top 32. Recap that we know the actual product that the customer purchased. The next step is to calculate the reciprocal of the actual product’s rank for each of \( r_1, \ldots, r_{10} \). For example, suppose for \( r_1 \), the actual purchased product is the second within the 32 returned products, then the score for \( r_1 \) is \( \text{score}_1 = 1/2 = 0.5 \). The score of the rewritten queries \( r_1, \ldots, r_{10} \) is then defined as \( \max \{ \text{score}_i \}_{i=1}^{10} \). Finally, the score for the query rewriting model is the average over all the source query scores.

We also use HIT@1 and HIT@16 as evaluation metrics. The HIT@16 metric is the percentage that the actual product is ranked within the first 16 products (the first page) when we search the rewritten query. And the HIT@1 metric is similarly defined.

### B.3 Training Details

We use the Fairseq (Ott et al., 2019) code-base with PyTorch (Paszke et al., 2019) as the back-end to implement all the methods. All the experiments are conducted using 8 NVIDIA V100 (32GB) GPUs.

For training a Transformer model from scratch, we adopt the Transformer-base (Vaswani et al., 2017) architecture. We use Adam (Kingma and Ba, 2015) as the optimizer, and the learning rate is chosen from \{3 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}\}. We use 4 heads for the multi-head graph attention mechanism, where the head dimension is set to be 128 (note that the Transformer-base architecture has embedding dimension 512).

For fine-tuning a BART model, we adopt the BART-base (Lewis et al., 2020) architecture. We use AdamW (Loshchilov and Hutter, 2019) as the optimizer, and the learning rate is chosen from \{3 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}\}. Similar to the training from scratch scheme, we adopt 4 heads, each with dimension 192, for the graph attention mechanism.

For both training-from-scratch and fine-tuning, please refer to\footnote{https://github.com/pytorch/fairseq/blob/master/examples/translation/README.md} Ott et al. (2019) for more details such as pre-processing steps and other hyper-parameters.

### C More Experimental Results

#### C.1 Analysis

- **BART vs. Transformer** Even though BART contains twice the number of parameters compared with Transformer (140M vs. 70M), models fine-tuned on BART yield lower MRR and HIT metrics (Table 1). One reason is that publicly available pre-trained models are pre-trained on natural language corpus, but queries are usually short and have distinct structures. This raises doubts on whether cur-
rent pre-trained models are suitable for the query domain. Indeed, the rich semantic information enables a much better BLEU score (32.9 vs. 28.2), but the MRR and HIT metrics suggest the fine-tuned models’ unsatisfactory performance.

Another reason is that in a conventional fine-tuning task, a task-specific head is appended to the pre-trained model, and the head usually contains only a small number of parameters. But in the query rewriting task, both the aggregation network and the graph attention mechanism contain a significant amount of parameters (about 10% of BART). This is problematic because in fine-tuning, the learning rate is usually small since nearly all the weights are supposed to be meaningful and should not change much. Yet, in our case, we need to properly train a large amount of randomly initialized parameters. Moreover, the aggregation network and the GAT are added inside the pre-trained model (more specifically, they are added to the BART encoder) instead of appended after BART. Essentially this nullifies the pre-trained parameters on the decoder side, imposing additional challenges to the fine-tuning task. Nevertheless, the BART+Aggregation model still outperforms the vanilla BART model, and the performance is further improved by adding the GAT (i.e., BART+Aggregation+Graph).

Training from scratch vs. fine-tuning Figure 5 plots the training and validation perplexity (ppl) of the training-from-scratch approach and the fine-tuning approach. From Figure 5a and Figure 5b, we can see that by employing the aggregation network, Transformer+Aggregation fits the data better and exhibits enhanced generalization. The training and validation ppls are further significantly improved by incorporating the graph attention mechanism, i.e., by using Transformer+Aggregation+Graph, we achieve even better performance.

Notice that in Figure 5c, BART+Aggregation outperforms BART+Aggregation+Graph in terms of training ppl, which is different from the training-from-scratch approach. As indicated by Figure 5d, BART+Aggregation shows clear sign of over-fitting. This is because even though pre-trained language models contain rich semantic information, much of it is considered “noisy” for query rewriting. Thus feature enhancement initiated by the graph attention mechanism is needed.

Model size vs. performance Figure 6 illustrates the relation between model size and performance, where we decrease the embedding dimension (correspondingly the FFNs’ hidden dimensions) and the number of layers. We can see that even with 1/8 of the parameters, model performance does not decrease much. Moreover, our model is more than 20% smaller than a BERT-base model (85M vs. 110M), rendering online deployment more than possible.

Query length vs. performance Figure 7 demonstrates model performance regarding length of the instant query. We can see that the BLEU score gradually decreases when the length increases. This is because long queries are often very specific (e.g., down to specific models or makes), making the rewriting task harder.

D Detailed Algorithm
Figure 5: Training and validation perplexity using Transformer and BART as backbone.

Figure 6: Model performance (in BLEU scores) vs. model size. The model size (in millions of parameters) are shown above the bars.

Figure 7: Query length vs. rewriting quality.

Algorithm 1: Context-aware query rewriting.

**Input:** $D$: dataset containing sessions; Initial parameters for the Transformer encoder and the Transformer decoder; Initial parameters for two graph attention mechanism (Eq. 2): $\text{GAT}_{t \rightarrow q}$, $\text{GAT}_{q \rightarrow t}$; Initial parameters for the aggregation network (Eq. 3); $K$: the number of updates on the session graph; $N$: the number of rewritten queries for each session.

**Output:** A list that contains $N$ generated queries for each session in the dataset.

Rewritten results: $\text{rewritten} = \{\}$

for each session in $D$ do

/* Encode input data. */
Compute source representation $H_s$ and history representation $U_h$ using the Transformer encoder;

/* Apply graph attention. */
Obtain initial representations $G^0_t$, $G^0_q$;

for $k = 1 \cdots K$ do

$G^k_t = \text{GAT}_{q \rightarrow t}(G^{k-1}_q, G^{k-1}_t)$;

$G^k_q = \text{GAT}_{t \rightarrow q}(G^{k-1}_t, G^{k-1/2}_q)$;

end

Set history representation $\{h_i\}_{i=1}^{N_t+N_h} = G^K_t \cup G^K_q$;

/* Apply aggregation network. */
Compute session representation $H_{\text{sess}}$ from $H_s$ and $\{h_i\}_{i=1}^{N_t+N_h}$ using Eq. 3;

/* Generate rewritten queries. */
Generate $N$ rewritten queries $\{q_i\}_{i=1}^N$ using the Transformer decoder and a beam search procedure;

$\text{rewritten} \leftarrow \text{rewritten} \cup \{q_i\}_{i=1}^N$;

end

**Output:** The rewritten queries.