Query2Vec: Learning Deep Intentions from Heterogenous Search Logs

ABSTRACT

The success of deep learning has been applied to many natural language processing applications such as language model, machine translation, parsing, sentiment analysis and so on. The recent improvement of neural language model such as Word2Vec improves accuracy of word embedding with much lower computational cost. The Word2Vec, however, is not properly designed to extract user intentions from search logs due to their sparseness and heterogeneity such as clicks, sessions, documents and so on. In this paper, we propose Query2Vec that simultaneously learns sparse clicks, sessions and textual documents to improve query embedding task and collection ranking task by extracting deep intentions from the long tail queries. We tested our model on real click data collected during 2014 from a commercial search engine in Korea.

1. INTRODUCTION

Neural networks learn high level of abstraction that consists of multiple hidden layers with non-linear activation to obtain new combination of features or representations. Beyond images and speeches, language also has been studied recently such as machine translation, sentiment analysis, and language modeling. Word embedding that extracts representative features of words and project them into vector spaces is an important language modeling problem. Due to high dimensionality of text, generally word embedding is used for pretraining. The neural network language model (NLDM) is based on N-gram model that receives back and front words of target word and calculate probability distribution of possible target words through projection layer and hidden layers. Recently, word2vec has been developed to improve accuracy with much lower computational cost by using shallow neural networks. One of the proposed algorithms, Skip-gram, learns five to ten contextual words given a word using 1-layer neural network since they are more likely close in word representation. Unlike frequency based models such as topic modeling, skip-gram projects (partial) sequential information into vector spaces so capturing syntactic or semantic knowledge from same position in sentences. The learned word vectors can perform inference by combining the vectors of words and finding the closest word of vector with the combined vector. For example, "Baseketball – MichaelJordan + Golf gives TigerWoods."

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Figure 1: Number of queries over grams during October 2014 following power-law distribution (left) and number of queries over document collections during 2014 (right)

The modern commercial search engines highly rely on their click data such as click-through bipartite and users’ relevance feedback. While highly clicked queries are easy to obtain intentions from the click data, low clicked queries are still challenging to rank. The figure 1(a) shows distribution of queries with respect to their grams collected from October 2014 in a commercial search engine. The low clicked queries are difficult to extract users’ intentions from the data and still many queries exist in the long tail.

In this paper, we proposed Query2Vec to extract users’ intention from search logs (especially for low clicked queries) using a shallow neural language model. Our model concatenates heterogeneous search logs such as clicks, session, and normal texts and train them differently with respect to the types. For example, Query2Vec learns click data whose input is query and output is a title sentence of clicked document to predict. By doing so, the models trains conditional probability \( P(w_{clicked}) \) to predict their contextual words given a query. Among the three types of search queries: informational queries, navigational queries, and transactional queries, we assume that Query2Vec more accurately distinguish transactional queries that reflect the intent of the user to perform a particular action such as how to purchase a car, how to visit Seoul from Melbourne, and opening time for restaurants.

To validate our method, we collected search logs during 2014 from a commercial search engine in South Korea. First, we tested Query2Vec for collection ranking task: when you query a name of restaurants to go, a search engine gives you directly a spot in the map. In vertical search engine, ranking correct verticals (e.g., scholar, web, images, video, maps) to a given query is a very important problem. The figure 1(b) shows a distribution of top-40 verticals and the number of queries matched to the vertical at first during 2014 from a commercial search engine in Korea. In general, our Query2Vec achieves 30.2% of top-1 classification result for collection ranking: 46.2% for high click queries and 23.4% for low
click queries. Moreover, we also generated syntactic and semantic inference pairs for Korean language as [12] did. Our concatenated Query2Vec achieves 30.2% for semantic inference and 2.8% that outperforms Skip-gram based models. A bit surprisingly, Query2Vec is a first attempt to embed queries into vector spaces for ranking task.

2. QUERY2VEC

To learn heterogeneous types of search logs, it is necessary for learning algorithm to deal with the heterogeneity. The simplest way to do so is to concatenate (1) learnt model parameters (model concatenation) or (2) input data (data concatenation) from the different sources of search logs. The model concatenation, however, does not give mutual relationship between the sources. Thus, we choose to concatenate input data and develop different learning algorithm for each source. The Figure 2 describes four different algorithms of Query2Vec: SkipGram with texts, QueryGram with queries, ClickGram with clicks, and sessionGram with sessions.

For textual data (Figure 2 (a)) such as sentences from Blogs, News articles, Web documents, etc, we simply used SkipGram algorithm from [12][13]. The purpose of learning language model [3][12] is for given a word to predict next or surrounding words in a sentence based on word representations. For simplicity, we use mathematical derivations from [7] that explains negative sampling more formally. When a corpus of word w is given with their context c (a set of context words w), we like to predict the context c given w. Usually, c is words that appear back and forth on w(t) such as w(t−1) and w(t+1). More formally, conditional probability P(c|w) and given corpus V, the goal is to maximize the corpus probability with the model parameters θ of p(c|w; θ).

\[
\arg \max_{\theta} \prod_{w \in V} p(c|w; \theta)
\]

where context words c_i are adjacent 2k words of the given word w, such as c = (w_k, w_k) where −k ≤ c ≤ k in [7]. The conditional probability p(c|w; θ) can be written using soft-max. Then, after taking log and switch the equation from product to sum:

\[
\arg \max_{\theta} \prod_{w \in V} p(c|w; \theta) \approx \arg \max_{\theta} \sum_{w \in V} \log p(c|w)
\]

\[
\approx \sum_{w \in V} \left( \log e^{c \cdot v_w} - \log \sum_{c \in C} e^{c \cdot v_w} \right)
\]

where \(\theta = v_w, v_c\) for w \(\in V, c \in C, i \in (1, \ldots, d)\). To compute the conditional probability p(c|w; θ) is very expensive due to the \(\sum_{c \in C} e^{c \cdot v_w}\). hierarchical softmax is used by reducing the complexity from V to \(\log(V)\). In the word2vec, they used a binary Hoffman tree, as it assigns short codes to the frequent words which results in fast training. [13] presents negative sampling as another alternative to the hierarchical softmax. It prevents all the vectors from having same value, by disallowing some (w, c) combinations for which their conditional probabilities must be low, i.e. pairs which are not in the training data. Usually, the number of such pairs called \(D'\) that are randomly sampled negative examples are \(K\) times larger than observed training data D, so for each \((c, w) \in D\) we construct \(k\) samples \((w, c_1), \ldots, (w, c_k)\). Thus, the objective function can be written:

\[
\arg \max_{\theta} \prod_{(w, c) \in D} \log \sigma(v_c \cdot v_w) + \prod_{(w, c) \in D'} \log \sigma(-v_c \cdot v_w)
\]

where \(\sigma = \frac{1}{1 + e^{−t}}\) and \(D \cup D'\) are entire corpus. Finally, in order to prune frequent words that might be less informative, they prune words appearing less than \(min \approx \frac{count}{freq}\) and each word with probability computed by \(P(w_i) = 1 - \sqrt{\frac{freq(w_i)}{t}}\) where t is a threshold (usually around 10−3) and \(freq(w_i)\) is the frequency of word \(w_i\). Please find [7] for detail derivations.

For queries (Figure 2 (b)), we propose QueryGram that regards a query as a sentence of words and train it as similarly as SkipGram does. Unlike SkipGram, we model p(q(t−1), q(t+1)|q(t); θ). Due to sparseness of queries (See Figure 1), however, QueryGram suffers from lack of sequential information; length of most queries are less than 5 and N-gram contains only \(N − 1\) sequential information. Though, QueryGram itself can be used for query auto-completion task [2].

To address the sparseness and deal with click data, we propose ClickGram whose input is a query and output is sentences of the clicked document given the query (Figure 2 (c)). For a given query q, ClickGram predicts clicked words instead of surrounding words in QueryGram. Thus, the objective is to model p(w(t−1), w(t), w(t+1)|q; θ). Our idea is very simple but effective to learn association between queries and clicked documents such as other click-through bipartite based methods [6][10].

At last, we propose SessionGram to learn intentions from session data (Figure 2 (d)) that concatenates a sequence of queries in a session and regards the concatenated queries as a sentence. By modeling p(q(s−1), q(s+1)|q(s); θ), given a query q, SessionGram can predict next sequence of queries such as query-reformulation [9].

With respect to the type of data in search logs, we suggested different types of learning algorithms. While each model has their own objective, we also concatenate them all together and generate a unified Query2Vec vector.

3. EXPERIMENTS

We collect search logs during 2014 from a commercial search engine in Korea [1]. The Table 1 shows statistics of our dataset: queries, blogs, news, sessions, clicks and their concatenation (e.g.,
Table 1: Dataset. We collected queries, blogs, news, sessions, clicks and their concatenation such as clicks + news, clicks + sessions, and clicks + session + news. Each dataset contains following number of sentences or queries, words that are distinct terms after filtering, and vectors that are final words after filtering the words less than 10.

<table>
<thead>
<tr>
<th></th>
<th>queries</th>
<th>blogs</th>
<th>news</th>
<th>sessions</th>
<th>clicks</th>
</tr>
</thead>
<tbody>
<tr>
<td>sentences</td>
<td>18,804</td>
<td>277</td>
<td>200,012</td>
<td>381</td>
<td>71,961</td>
</tr>
<tr>
<td>words</td>
<td>67,554</td>
<td>933</td>
<td>1,197,663</td>
<td>722</td>
<td>578,234</td>
</tr>
<tr>
<td>vectors</td>
<td>434,421</td>
<td>1,671,156</td>
<td>597,678</td>
<td>709,257</td>
<td>923,490</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>clicks</th>
<th>news</th>
<th>sessions</th>
<th>clicks + news</th>
<th>clicks + sessions</th>
<th>clicks + session + news</th>
</tr>
</thead>
<tbody>
<tr>
<td>sentences</td>
<td>158,630,233</td>
<td>69,668,652</td>
<td>168,630,233</td>
<td>1,507,141,211</td>
<td>998,098,877</td>
<td>1,577,232,777</td>
</tr>
<tr>
<td>words</td>
<td>1,206,287</td>
<td>1,444,757</td>
<td>1,724,672</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We joined the Query2Vec from the Blogs and the News. In order to obtain syntactic set of 14, 476, 181 queries and 271, 973, 962 clicked documents from the Blogs and the News. In order to obtain the query, word) pairs, we joined the <query, document> with <document, words in a title> pairs. Then, every sentences are tagged using a Korean morphological tagger such as [11]. After filtering, we obtain words and Query2Vec filtered out again the words occurring less than 10 and generates vectors.

For each dataset, we compared following algorithms:
- SGblog or SGnews: SkipGram with Blog titles or News titles.
- QGquery: QueryGram with queries.
- SSession: SessionGram with sessions.
- CG: ClickGram with clicks.
- cGqv: a concatenated gram of ClickGram with clicks and SkipGram with news.
- cGv: a concatenated gram of ClickGram with clicks and SessionGram with sessions.
- cGv: a concatenated gram of ClickGram with clicks and SkipGram with news.

Table 1 shows results of query embedding for the generated semantic and syntactic pair sets. For single source of data, CGclick outperforms other algorithms on both syntactic (16.4%) and semantic set (12.3%). In our experiment, ClickGram algorithm is the most effective algorithm for learning association between queries and clicks compared to other baselines such as either QueryGram or SkipGram with clicks. For concatenation of multiple sources of data, CGv.na that combines clicks, news, and sessions outperforms on both tasks (22.3% for syntactic 14.8% for semantic) than other concatenation models with two sources such as CGv.na and CGv. The more concatenation of different types of sources, the better accuracy of semantics and syntactics we obtain.

3.1 Query Embedding

To evaluate how useful the learned vectors of queries are, we follow the pair inference task in [12] given a relational pair of two words (w1, w2) and a new word (w3), predicting a word (w4) that is related with the given word (w3). For example, Paris and France has a relationship of city-country as similarly as Seoul is a city of country Korea. [12] released a public test set for semantic and syntactic pair inference task. However, there are no Korean dataset.

We generate our own Korean semantic and syntactic pairs.

For semantic pair set, we collected knowledge databases from a Korean commercial search engine [1]. Among many categories of databases such as cooking, game, movie, etc, we only choose people database and generate relationship pairs of people with their family members, jobs, movies, and on-airs. We collected total 1, 183, 824 semantic pairs and finally obtained 512, 121 after filtering out non-covered pairs that don’t exist in our model vectors. For example, a relationship of job has a pair of <ParkGuenHye, Politician> and <Psy, Singer>.

For syntactic pair set, we extracted Korean grammar rules from Wiktionary[1] and Korean dictionary[2]. Wiktionary consists of approximately 3.5 million entries in 172 language editions. We extracted total 3, 381, 425 pairs of transformation and inflection rules from the Korean Wiktionary dataset such as <그다, 크니가> and <가다, 가니가>. In addition, we extracted 1, 194, 560 pairs of synonym and antonym rules from the Korean dictionary such as <한, 오른, and <현, 진>. After filtering out non-covered pairs by the vectors of our models, we obtained 585, 822 pairs of transformation and inflection rules 451, 487 pairs of synonym/antonym rules.

Table 2: Accuracy of Query2Vec algorithms for syntactic set (left) and semantic set (right). The semantic set consists of 512, 121 people related pairs such as family, job, movie and onair relationships. The syntactic set consists of 585, 822 pairs of transformation/inflection rules and 451, 487 pairs of synonym/antonym rules. Each algorithm is trained with 200 dimension of hidden layer and 10 minimum word count.

3.2 Collection Ranking

To evaluate how effectively Query2Vec reflect users’ intentions, we measure accuracy of ranking for collections. For a given query, ranking relevant collections is very challenging task in vertical search engine. For example, if a user sends a query “how to visit Seoul station”, then map collection directing to the Seoul station should be ranked at first. Our goal is not to outperform the existing collection ranking but providing additional vectors of queries to improve the performance especially for long tail queries.

We trained our algorithms in a same environment setting: dimension of first hidden layer is 200, minimum count of words are 10, number of threads are 20, and a machine is single 24-core xeon. The Table 2 shows results of query embedding for the generated semantic and syntactic pair sets. For single source of data, CGclick outperforms other algorithms on both syntactic (16.4%) and semantic set (12.3%). In our experiment, ClickGram algorithm is the most effective algorithm for learning association between queries and clicks compared to other baselines such as either QueryGram or SkipGram with clicks. For concatenation of multiple sources of data, CGv.na that combines clicks, news, and sessions outperforms on both tasks (22.3% for syntactic 14.8% for semantic) than other concatenation models with two sources such as CGv.na and CGv. The more concatenation of different types of sources, the better accuracy of semantics and syntactics we obtain.
By 5-fold cross validation using a SVM classifier, the Table 3 shows top-1 classification due to the vulnerability of collection labels.

**Table 3:** Classification accuracy of Query2Vec algorithms on three different partition of queries: whole query set, high clicked queries (20%), and low clicked queries (80%). The accuracy is calculated by 5-fold cross validation using a SVM classifier.

<table>
<thead>
<tr>
<th></th>
<th>random</th>
<th>SGquery</th>
<th>SGlog</th>
<th>SGnews</th>
<th>SGsession</th>
<th>QGclick</th>
</tr>
</thead>
<tbody>
<tr>
<td>whole</td>
<td>10.9%</td>
<td>31.1%</td>
<td>30.2%</td>
<td>30.5%</td>
<td>28.0%</td>
<td>30.7%</td>
</tr>
<tr>
<td>high</td>
<td>15.3%</td>
<td>36.9%</td>
<td>33.8%</td>
<td>34.1%</td>
<td>30.4%</td>
<td>46.2%</td>
</tr>
<tr>
<td>low</td>
<td>8.4%</td>
<td>24.5%</td>
<td>26.4%</td>
<td>26.7%</td>
<td>24.7%</td>
<td>23.4%</td>
</tr>
</tbody>
</table>

With the vectors of queries, we conducted 5-fold validation tests using a SVM classifier. The Table 3 shows top-1 classification results: we partitioned our dataset into three parts: whole query set, high clicked queries (approximately 20%) and low clicked queries (80%) of accuracy on whole dataset. However, QGclick performs almost 46.2% on high clicked queries since the collection labels are mostly decided by the click data. Unlike high clicked queries, most algorithms on low queries that exist in long tail perform similarly around 25%.

The Figure 3 shows a visualization of our query embedding in 2-d plot using t-Distributed Stochastic Neighbor Embedding (t-SNE) [18]. We only projected 1,000 words that are randomly chosen from the vectors. Figure 3 (a) is an output of QueryGram trained by queries that similar queries are grouped together (circles) such as cosmetic brands, English words, and geological words. Each group of queries represents a semantically or syntactically similar cluster that reflects a user’s intention. Figure 3 (b) is an output of ClickGram trained by clicks. Although queries that have similar intentions are clustered, the colors in the cluster are so diverse meaning that our collection labels are not accurate enough to evaluate rankings of such NDCG [5] or Kendall’s tau [14].

**Figure 3:** Visualization of Query-Embedding using t-SNE [18]. (a) QueryGram with clicks and (b) ClickGram with clicks colored by types of collections. Queries that have similar intentions are clustered well.

4. **CONCLUSION**

5. **REFERENCES**