Query2Vec: Learning Intentions from Heterogenous Search Logs

Dongyeop Kang
Naver Labs
dongyeop.kang@navercorp.com
September 27, 2016

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 sparsness 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.

Introduction
Neural networks learn high level of abstraction that consists of multiple hidden layers with non-linear activation to obtain a new combination of features or representations [8]. Beyond image and speech recognition, a language domain also has been studied recently such as machine translation [17], sentiment analysis [15], and language modeling [16]. Word embedding that extracts representative features of words and projects them into vector spaces is an important language modeling problem. Due to high dimensionality of text, word embedding is generally used for pretraining. The neural network language model (NNLM) [5] is based on N-gram model that receives back and front words of target words and calculate the probability distribution of possible target words through a projection layer and hidden layers. Recently, word2vec [12][13] has been developed to improve accuracy with much lower computational cost by using shallow neural networks. One of the proposed algorithms, Skipgram, learns five to ten contextual words given a word using an 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 the same position in sentences. The learned word vectors can perform inference by combining the vectors of words and finding the word of the vector closest to the combined vector. For example, 

```
Baseketball − MichaelJordan + Golf gives TigerWoods.
```

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)

Modern commercial search engines highly rely on their click data such as click-through bipartite [6] and users’ relevance feedback [10]. 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 distributions of queries with respect to their grams collected from October 2014 in a commercial search engine. The low clicked queries that are very important to evaluate search engines’ performance are difficult to extract users’ intentions from the data and still many queries exist in the long tail.

Here, 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, sessions, and normal texts and trains them with different strategies dependent on their types. For example, Query2Vec learns click data whose input is query and output is a title sentence of clicked document to be predicted. This allows the models to train conditional probability $P(w_{clicked}|q)$ for predicting their contextual words given a query. Among the three types of search queries [4] – informational queries, navigational queries, and transactional queries, we assume that Query2Vec more accurately distinguishes 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 a collection ranking task: when you query the name of a restaurant to go, a search engine gives directly you a
spot in the map. In a 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 the commercial search engine. 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 14.8% for semantic inference and 22.3% for syntactic inference which outperforms Skip-gram based models. A bit surprisingly, Query2Vec is a first attempt to embed queries into vector spaces for ranking task.

**Query2Vec**

To learn heterogeneous types of search logs, it is necessary for learning algorithms to deal with the heterogeneity. The simplest way is to concatenate (1) learned 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
where $\sigma$ can be written:

each (sampled negative examples are $K$ not in the training data. Usually, the number of such pairs called $D$ combinations for which their conditional probabilities must be low, i.e. pairs which are

softmax. It prevents all the vectors from having same value, by disallowing some (fast training. \cite{13} presents negative sampling as another alternative to the hierarchial binary Ho is used by reducing the complexity from $V$ probability $p$ model $p$ threshold (usually around $10^{-5}$)

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

where context words $c_i$ are adjacent $2k$ words of the given word $w_i$ such as $c = (w_{i+k})$ where $-k \leq c_i \leq k$ in \cite{7} The conditional probability $p(c|w; \theta)$ can be written using soft-max. Then, after taking a log function and switch the eq 1 from product to sum:

$\arg \max_{\theta} \prod_{(w,c) \in D} p(c|w; \theta) \approx \arg \max_{\theta} \sum_{(w,c) \in D} \log p(c|w)$

$\approx \sum_{(w,c) \in D} (\log e^{v_c \cdot v_w} - \log \sum_{c' \in C} e^{v_c \cdot v_{w'}})$

where $\theta$ is $v_c, v_{w_i}$ for $w \in V, c \in C,$ and $i \in (1, \ldots, d).$ To compute the conditional probability $p(c|w; \theta)$ is very expensive due to the $\sum_{c' \in C} e^{v_c \cdot v_{w'}}$, the hierarchical soft max is used by reducing the complexity from $V$ to $\log_2 (V)$. In the word2vec, they used a binary Hoffman tree, as it assigns short codes to the frequent words which result in fast training. \cite{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:

$\arg \max_{\theta} \prod_{(w,c) \in D} \log \sigma(v_c \cdot v_{w_i}) + \prod_{(w,c) \in D'} \log \sigma(-v_c \cdot v_{w})$

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 - count times and each word with probability computed by $P(w_i) = 1 - \sqrt{\frac{t}{f_{\text{freq}}(w_i)}}$ where $t$ is a threshold (usually around $10^{-5}$ and $f_{\text{freq}}(w_i)$ is the frequency of word $w_i$. Please find \cite{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 $p(q(t-1), q(t+1)|q(t); \theta).$ 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 \( p(w(t-1), w(t), w(t+1)|q; \theta) \).

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 concatenate a sequence of queries in a session and regards the concatenated queries as a sentence. By modeling \( p(q(s-1), q(s+1)|q(s); \theta) \), given a query \( q \), SessionGram can predict a next sequence of queries such as query-reformulation [9].

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

Experiments

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 out 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>18,804,277</td>
<td>200,012,381</td>
<td>71,961,581</td>
<td>10,000,000</td>
<td>39,239,805</td>
</tr>
<tr>
<td>words</td>
<td>67,554,933</td>
<td>1,197,663,722</td>
<td>578,234,313</td>
<td>69,328,038</td>
<td>1,970,343</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>
<tr>
<td>clicks + news</td>
<td>158,630,233</td>
<td>96,668,652</td>
<td>168,630,233</td>
<td>1,577,232,777</td>
<td></td>
</tr>
<tr>
<td>clicks + sessions</td>
<td>1,507,141,211</td>
<td>998,098,877</td>
<td>1,577,232,777</td>
<td>1,724,672</td>
<td></td>
</tr>
<tr>
<td>clicks + session + news</td>
<td>1,206,287</td>
<td>1,448,757</td>
<td>1,724,672</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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., clicks + news, clicks + sessions, clicks + session + news). From the raw data, we filtered out each dataset as follows: Blogs whose quality level is regarded as “bad” are filtered out. News articles that are classified as a spam are filtered out. Queries that occur less than 10 and clicks that occur less than 10 are filtered out. Clicks whose clicked documents are only from the blogs or the news articles are used for fair comparison. For example, the 39,239,805 clicks consist of 14,476,181
queries and 271,973,962 clicked documents from the Blogs and the News. In order to obtain \( \text{query, word} \) pairs, we joined the \( \text{query, document} \) with \( \text{document, words in a title} \) pairs. Then, every sentence is tagged using a Korean morphological tagger such as [11]. After filtering, we obtain \text{words} and \text{Query2Vec} filtered out again the words occurring less than 10 and generate vectors.

For each dataset, we compared following algorithms:

- \( \text{SG}_{\text{blog}} \) or \( \text{SG}_{\text{news}} \): \text{SkipGram} with Blog titles or News titles.
- \( \text{QG}_{\text{query}} \): \text{QueryGram} with queries.
- \( \text{SG}_{\text{session}} \): \text{SessionGram} with sessions.
- \( \text{CG}_{\text{c}} \): ClickGram with clicks.
- \( \text{cG}_{\text{c,n}} \): a concatenated gram of ClickGram with clicks and \text{SkipGram} with news.
- \( \text{cG}_{\text{c,s}} \): a concatenated gram of ClickGram with clicks and \text{SessionGram} with sessions.
- \( \text{cG}_{\text{c,n,s}} \): a concatenated gram of ClickGram with clicks, \text{SkipGram} with news, and \text{SessionGram} with sessions.

We evaluated the algorithms of \text{Query2Vec} on query embedding task and collection ranking task.

**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 \((w_1, w_2)\) and a new word \((w_3)\), predicting a word \((w_4)\) that is related with the given word \((w_3)\). For example, \text{Paris} and \text{France} has a relationship of city-country as similarly as \text{Seoul} is a city of country Korea. [12] released a public test set for a semantic and syntactic pair inference task. However, there is no Korean dataset. We generate our own Korean semantic and syntactic pairs.

For a semantic pair set, we collected knowledge databases from the 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-air. 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 \( \text{ParkGuenHye, Politician} \) and \( \text{Psy, Singer} \).

For a syntactic pair set, we extracted Korean grammar rules from Wiktionary1 and Korean dictionary2. 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 \( \text{크다, 크니가} \) and \( \text{가다, 가니가} \). In addition, we extracted 1,194,560 pairs of synonym and antonym rules from the Korean dictionary such as \( \text{원, 오른} \) and \( \text{현, 전} \). 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 and antonym rules.

We trained our algorithms in a same environment setting: the dimension of first hidden layer is 200, the minimum count of words is 10, the number of threads is 20,

---

1http://meta.wikimedia.org/wiki/Wiktionary
2http://krdic.naver.com/
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.

<table>
<thead>
<tr>
<th></th>
<th>Syntactic</th>
<th>Semantic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Auto.</td>
<td>Syno.</td>
</tr>
<tr>
<td>QG_{query}</td>
<td>9.8%</td>
<td>13.9%</td>
</tr>
<tr>
<td>SG_{blog}</td>
<td>10.1%</td>
<td>13.3%</td>
</tr>
<tr>
<td>SG_{news}</td>
<td>17.2%</td>
<td>15.3%</td>
</tr>
<tr>
<td>SG_{session}</td>
<td>19.8%</td>
<td>11.2%</td>
</tr>
<tr>
<td>CG_{click}</td>
<td>20.2%</td>
<td>12.8%</td>
</tr>
<tr>
<td>cG_{c,n}</td>
<td>23.1%</td>
<td>17.5%</td>
</tr>
<tr>
<td>cG_{c,s}</td>
<td>11.8%</td>
<td>14.7%</td>
</tr>
<tr>
<td>cG_{c,n,s}</td>
<td>24.1%</td>
<td>20.5%</td>
</tr>
</tbody>
</table>

and a machine is single 24-core xeon. The Table 2 shows the results of query embedding for the generated semantic and syntactic pair sets. For the single source of data, CG_{click} outperforms other algorithms on both the syntactic (16.4%) and the semantic set (12.3%). In our experiment, ClickGram algorithm is the most effective algorithm for learning associations between queries and clicks compared to other baselines such as either QueryGram or SkipGram with clicks. For concatenation of multiple sources of data, cG_{c,n,s} 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 cG_{c,n} and cG_{c,s}. The more concatenation of different types of sources, the better accuracy of semantics and syntactics we obtain.

Collection Ranking

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

We first collected collection ranks of the queries during 2014 from the commercial search engine, and choose only top-40 collection labels out of 287 such as web, blogs, news, map, and so on. (See Figure 1(b)). Even though the collections are scrapped from the search engine, they are not correct answer set because the search engine also suffers from the long tail issues and highly relies on click data. Unlike general evaluation metrics for ranking such NDCG [5] or Kendall’s tau [14], we simply measure top-1 classification result due to the vulnerability of collection labels.

With the vectors of queries, we conducted 5-fold cross validation tests using a SVM
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>SG_query</th>
<th>SG_blog</th>
<th>SG_news</th>
<th>SG_session</th>
<th>QG_click</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>

classifier. The Table [3] shows top-1 classification results: we partitioned our dataset into three parts: whole query set, high clicked queries and low clicked queries (approximately 20% and 80% of whole query set, respectively). Except random algorithm, most of algorithms perform approximately 30% of accuracy on whole dataset. However, QG\_click performs almost 46.2% on high clicked queries since the collection labels are mostly decided by the click data. Unlike high queries, most algorithms on low queries that exist in long tail perform similarly around 25%.

Figure 3: Visualization of Query-Embedding using t-SNE [18]. QueryGram with clicks. Queries that have similar intentions are clustered well.

The Figure 3 and Figure 4 visualizes 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 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 a syntactically similar cluster that reflects user’s intention. Figure 4 is an
ClickGram with collections

Figure 4: Visualization of Query-Embedding using t-SNE. ClickGram with clicks colored by types of collections. Queries that have similar intentions are clustered well.

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 collection and highly biased to a few top collection such as web, blg, cafe.

Conclusion

In this paper, we propose Query2Vec to learn intentions from sparse and heterogeneous queries logs. From the empirical evaluation on real query logs from NAVER search engine, we could obtain performance improvement on query embedding and collection ranking. In the future, we will import Query2Vec on large scales of search logs using multiple GPUs and CPUs.

References


