A Neural Language Model for Query Auto-Completion

Dae Hoon Park*
Huawei Research America
Santa Clara, CA, USA
daehpark@gmail.com

Rikio Chiba
Yahoo Inc.
Sunnyvale, CA, USA
rchiba@yahoo-inc.com

ABSTRACT

Query auto-completion (QAC) systems suggest queries that complete a user's text as the user types each character. Such queries are typically selected among previously stored queries, based on specific attributes such as popularity. However, queries cannot be suggested if a user's text does not match any queries in the storage. In order to suggest queries for previously unseen text, we propose a neural language model that learns how to generate a query from a starting text, a prefix. Specifically, we employ a recurrent neural network to handle prefixes in variable length. We perform the first neural language model experiments for QAC, and we evaluate the proposed methods with a public data set. Empirical results show that the proposed methods are as effective as traditional methods for previously seen queries and are superior to the state-of-the-art QAC method for previously unseen queries.

KEYWORDS

query auto-completion; neural language model; recurrent neural network

ACM Reference format:
Dae Hoon Park and Rikio Chiba. 2017. A Neural Language Model for Query Auto-Completion. In *Proceedings of SIGIR '17, Shinjuku, Tokyo, Japan, August 07-11, 2017,* 4 pages.
DOI: 10.1145/3077136.3080758

1 INTRODUCTION

When a user types text into a search box, most major search engines instantly suggest a list of queries that complete the user's text. Queries are suggested for each time a user types a character, in general, and this service is called Query Auto-Completion (QAC). Since the start of the service, QAC has been heavily used by users and has influenced the search results greatly [1]. The most common approach for QAC is to leverage wisdom of the crowd [1, 16], for example, MostPopularCompletion (MPC) [1], where likelihood of a query is estimated from the crowd's past queries that start with the user's text.

However, such traditional QAC systems are not adequate for queries that are not present in the storage. When there are no queries that start with a user's rare prefix, queries cannot be suggested because they do not exist in the storage. For example, let's suppose a prefix "places to go in Tokyo with " is given, and our storage indeed does not contain queries completing the prefix. Then, the traditional system will not suggest queries such as "places to go in Tokyo with family". The limitation of the traditional systems becomes more problematic for search engines that receive a relatively small number of queries, such as academic search.

*This work was done while at Yahoo Research.

Table 1: Top suggested queries by our neural language model trained on a public data set [13] collected in 2006. Suggestions with scores lower than the threshold are excluded. Phrases such as "afraid of the dead" and "afraid of the dog" and all prefixes do not exist in the data. Note that there is also a low-quality suggestion "when is a good time to buy a lyrics."

when is a good time to buy a
when is a good time to buy a **house**
when is a good time to buy a **home**
when is a good time to buy a **lyrics**
when is a good time to buy a **car**
why am i afraid of
why am i afraid of **the dark**
why am i afraid of **the dead**
why am i afraid of **the dog**
president donald
president donald **trump**

In this work, we study the problem of QAC especially for previously unseen queries. Specifically, we try to predict a query q for a prefix r. In order to overcome the traditional system's limitation, we propose a neural language model that can *generate* queries for a prefix based on their likelihood $p(q|r)$. Unlike the common approaches, our recurrent neural network (RNN)-based language model encodes variable-length prefix in a fixed-length state vector, and it predicts the most probable completing queries for the state. Output examples of our proposed model are shown in Table 1. The prefixes are not present in the data, so MPC cannot suggest any queries. However, our generative model can suggest reasonable queries. The main research questions we study are:

- Can language modeling approach perform as good as the traditional approach (MPC [1]) for queries that occurred in the past?
- How well can the proposed model predict queries that have never occurred in the past?

2 RELATED WORK

Query auto-completion has been studied in various directions, including context-aware and personalized QAC [1, 15] and time-sensitive QAC [16]. Those directions are orthogonal to our direction where we focus on generating queries for unseen queries, so we do not explore those directions in this work. A recent comprehensive survey on QAC is available in [5]. Most of the previous approaches rely on the selection-based method, which selects query candidates that are submitted by users as a whole, so they fail for prefixes that have never occurred. Vargas *et al.* [19] studied term-by-term QAC for mobile search using a query-term graph, but their goal is different from ours in that they do not predict a whole query and their prediction is in word-level while ours is in character-level. Szpektor *et al.* [18] studied to leverage templates to recommend long-tail queries, but their approach is limited to query suggestion where they suggest a set of related queries after they receive a full query. Mitra and Craswell [12] studied the most similar problem, QAC for rare prefixes, but their approach is quite different from ours. They collect popular n-gram suffix words from query log to synthesize them with a user's text, and they re-rank those candidates using a neural network-based model. However, instead of synthesizing existing text, we generate candidates by a character-level language model, meaning that our model is fully generative and more natural.

3 METHODS

The common approach (MPC) is not able to suggest a user's intended query if the query does not exist in the storage. To solve this problem, we propose to employ a language model that can generate a query given any prefixes even when they have never occurred before. A language model is defined as a probability distribution over a sequence of words $w_1, \dots, w_m$:

$$p(w_1, \dots, w_m) = \prod_{j=1}^{m} p(w_j | w_1, \dots, w_{j-1}). \quad (1)$$

However, due to the curse of dimensionality, it is difficult to reliably estimate a word's probability given all preceding words, $p(w_j | w_1, \dots, w_{j-1})$. One common approach is an n-gram model that makes n-th order Markov assumption, where $p(w_j | w_1, \dots, w_{j-1})$ is estimated as $p(w_j | w_{j-n}, \dots, w_{j-1})$, with smoothing techniques. Meanwhile, another approach that learns a distributed representation for words (embedding) has been shown to be effective for the dimensionality problem [2]. A language model that employs neural networks to resolve the dimensionality problem with a distributed representation of text is often called a neural language model. As neural language models have been reported to perform well [2, 10, 11], we follow this direction.

3.1 Neural Language Model for QAC

In QAC, we predict the intended query given a prefix, assuming the intended query always starts with the given prefix, as in most QAC work [5]. Thus, for an intended query q with texts $t_1, \dots, t_i, t_{i+1}, \dots, t_m$ where $t_1, \dots, t_i$ is the prefix r, we estimate query likelihood $p(q|r)$ by a query language model defined as

$$p(q|r) = p(t_{i+1}, \dots, t_m | t_1, \dots, t_i) = \prod_{j=i+1}^{m} p(t_j | t_1, \dots, t_{j-1}) \quad (2)$$

where each text t_k may be a word or a character.

Recurrent Neural Network. Reliably estimating $p(t_j | t_1, \dots, t_{j-1})$ is difficult because the number of possible prefixes increases exponentially as j increases. A recurrent neural network (RNN) can be a nice solution because the information from *all previous texts* can be represented as a low-dimensional state vector. In general, the distance between state vectors indicates the distance between information of the input texts. Thus, RNN can be useful to predict queries for an unseen prefix, especially when the unseen prefix is similar to some seen prefixes. A vanilla RNN has a vanishing gradient problem in practice, so we adopt Long Short-Term Memory (LSTM) [7] that augments RNN with a memory cell vector.

Character-level Neural Language Model. In the literature, neural language models are often in word-level where inputs or outputs are words [2, 8, 10, 11]. However, these architectures cannot handle Out-Of-Vocabulary (OOV) words well since they cannot interpret or predict OOV words. This issue becomes more problematic for domains with large vocabularies such as Web queries and Finnish text [9]. In addition, we want to leverage the *incomplete word* that often comes at the end of a user text. Thus, subwords such as characters and morphemes are more appropriate units for our problem. Morphemes require additional pre-processing [4], so we focus on a character-level language model, where both inputs and outputs are characters.

Word-embedded Space Character. Although character-level language models learn syntactic aspects of text well, it is difficult for

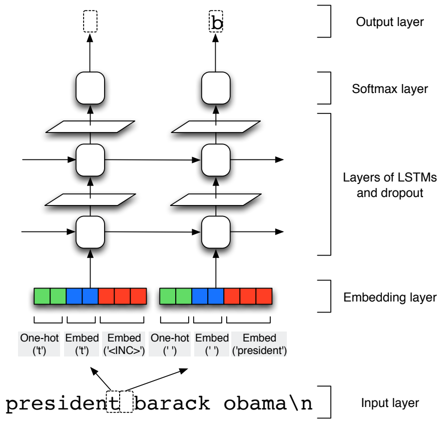

Figure 1: Architecture of our language model for an example query where '\n' indicates the end of the query. Green cells contain one-hot encoded vectors of characters, blue cells contain character-embedded vectors, and red cells contain word-embedded vectors.

them to learn semantic aspects in words from characters. We therefore incorporate word information into characters. To the best of our knowledge, there exists only one neural language model that merges words and characters to predict a text in character-level [3]. Their network contains different RNN cells for characters and words, and it jointly learns words and characters. Instead, we propose a simple yet effective method that can combine words and characters in a character unit.

Intuitively, when we process a prefix character by character, information of a complete word is only available when we reach the word boundary such as ' '. We thus pick up the word information at the word boundary. It can be easily implemented by embedding an incomplete word token '⟨INC⟩' to each character that is not ' ', and embedding a complete word only to ' '. We embed an unknown word token '⟨UNK⟩' to ' ' if the complete word does not appear at least K (=5 in this work) times in the query log, so that it can handle previously unseen words and save the number of parameters. In addition to the word-embedded vectors, we also place each character's one-hot encoded vector and embedded vector to combine the original input character and its distributed representation.

Architecture. The architecture of our neural language model is depicted in Figure 1. For each query, we map each character to the concatenated vector. The vector is passed to two layers of LSTMs, and the output character's probability is estimated by applying a softmax function to the output of the final LSTM layer:

$$p(t_{j+1} = c | t_1, \dots, t_j) = \frac{\exp(\boldsymbol{h}_j \cdot \boldsymbol{w}^c_{out} + b^c_{out})}{\sum_{c' \in V} \exp(\boldsymbol{h}_j \cdot \boldsymbol{w}^{c'}_{out} + b^{c'}_{out})}. \quad (3)$$

where $\boldsymbol{h}_j$ is a hidden state vector from the LSTM cell, $\boldsymbol{w}^c_{out}$ is an output weight vector's column for c, b^c_{out} is an output bias vector's value for c, and V is a set of all unique characters. To prevent over-fitting, we employ a dropout layer to the output of each LSTM cell [6]. Unlike general neural language models, where a model is trained on a single long sequence of text, we train the model on a set of query sequences by maximizing the following log-likelihood:

$$L = \sum_{q \in Q} \sum_{j=1}^{|q|-1} \log p(t_{j+1} | t_1, \dots, t_j) \quad (4)$$

where Q is a set of all queries in data, $|q|$ is the number of characters in a query q, and it is done by backpropagation through time [11]. Gradients are clipped to prevent gradient explosion.

3.2 Generating Query Candidates

Once we have a language model that can compute a character's likelihood such as $p(t_{j+1}|t_1, ..., t_j)$, we can generate query candidates with it. For example, we search for the most probable complete query from a prefix, character by character, until the path meets the end of the query, *i.e.*, '\n' in our query format. Each added character multiplies its probability to the path's probability to compute the complete query's probability as in equation (2). However, such greedy approach is fast but does not guarantee to find the optimal query, and it is too expensive to go through exponential number of all possible paths. Therefore, we use a beam search that keeps γ paths that are local optima. A simple version of our algorithm is described in Algorithm 1 where the subscript -1 denotes the last character of a variable. It can be easily extended to handle word embedding by remembering the position of the last word boundary for each path. Also, in practice, log-probability is computed to avoid underflow.

Algorithm 1: Query Candidate Generation

```
Input: A prefix $r$, a beam width $\gamma$, and a maximum suffix length $\delta$.
Output: Top $\gamma$ query candidates $\Sigma$.
// We assume that there is a function feedCharacter(char, state, suffix, prob, $\gamma$)
  that returns $\gamma$ character candidates that are most likely to come next if we add
  char in the current state ($\pi$). Returned candidates have updated suffix ($\omega$), $\pi$,
  and prob ($\lambda$);
$\pi$ = getInitialState(); // initialize LSTM cells;
for $i$=0 to $|r| - 2$ do
    $\pi$ = getNextState($r_i$, $\pi$); // feed $i$-th character to get the next state;
end
$\Sigma$ = feedCharacter($r_{-1}$, $\pi$, “”, 1.0, $\gamma$); // feed the last character in $r$;
repeat
    $\Sigma' = \Sigma; \Sigma = \{\}$ ;
    for $\sigma \in \Sigma'$ do
        if $\omega^\sigma$ ends with '\n' or has $\delta$ characters then
            $\Sigma = \Sigma \cup \{\sigma\}$ ;
        else
            $\Sigma = \Sigma \cup$ feedCharacter($\omega^\sigma_{-1}$, $\pi^\sigma$, $\omega^\sigma$, $\lambda^\sigma$, $\gamma$);
        end
    end
    $\Sigma$ = getNLargest($\Sigma$, $\gamma$); // keep only $\gamma$ candidates;
until $\forall \sigma \in \Sigma$ ( $\omega^\sigma$ ends with '\n' or has $\delta$ characters );
return $\Sigma$
```

4 EXPERIMENT DESIGN

Our language model is learned from a past query log (background data) and validated and tested by validation and test data. In the validation and test data sets, we follow the setting in [15], where we generate all possible prefixes for each query impression. We retain prefixes only if they contain at least one complete word as in [12]. For each prefix, we use our learned model to generate candidates as in Section 3.2. The prediction is evaluated by comparing with the original query. Specifically, we employ the standard metric Mean Reciprocal Rank (**MRR**) and a new metric Partial-matching MRR (**PMRR**), which are defined as

$$\text{MRR} = \frac{1}{|P|}\sum_{p\in P}\frac{1}{\text{r}_p}, \qquad \text{PMRR} = \frac{1}{|P|}\sum_{p\in P}\frac{1}{\text{pr}_p} \tag{5}$$

where $|P|$ is the number of all prefixes, r_p is the rank of the original query among the candidates for the prefix p, and pr_p is the rank of the first candidate that is the same as the original query or that extends the prefix by one or more complete words.[1] Note that $\text{pr}_p \le \text{r}_p$. In MRR, only original query is regarded relevant, but in PMRR, the candidates that partially match the original query are also regarded relevant. Thus, it can be useful to measure performance for an additive QAC that becomes more common in mobile devices [19]. We perform statistical significance test with a paired t-test ($p = 0.01$) and report it in Section 5 whenever needed.

We prepare several baselines including the common traditional approach MostPopularCompletion (**MPC**) [1]. We also experiment with character n-gram model (**Char. n-gram**) which we define as

$$p(q|r) = p(t_{i+1}, ..., t_m|t_1, ..., t_i) = \prod_{j=i+1}^{m} p(t_j|t_{j-n}, ..., t_{j-1}) \tag{6}$$

where $p(t_j|t_{j-n}, ..., t_{j-1})$ is estimated by its maximum likelihood based on frequency in background data. Its query candidates are generated by our method in Section 3.2. The closest work by Mitra and Craswell [12] is also implemented as a baseline. It leverages convolutional neural network (CLSM [14]) and LambdaMART[20] to rank candidates that include MPC-generated candidates and synthetic candidates. The ones using 10K and 100K synthetic candidates (**Mitra10K+MPC+λMART** and **Mitra100K+ MPC+λMART**) are used as in [12]. Parameters are tuned for maximum performance.

Our proposed language model has several versions. We compare the language model not using word-embedded character space (**NQLM**) with the ones using it (**+WE**). We test models with a small network using 512 hidden LSTM units (**NQLM(S)**) and a large network using 1,536 units (**NQLM(L)**).[2] We also append our language model-generated candidates to the end of MPC candidates, if there are any, and we add a symbol **+MPC**. As done in [12], we also employ LambdaMART and the same features, except that CLSM scores are replaced by our NQLM scores, to re-rank candidates (**+λMART**). For the recurrent neural network, we set the size of character and word-embedded vectors to 32 and 300. We initialize the word-embedded vector with the public pre-trained vector trained on Google News data set with 100 billion words, and the vectors are learned with our data set afterwards. Dropout rate is set to 0.25, and the models are trained for 5 to 10 epochs. The beam width γ in Section 3.2 is set to 10.

The public AOL query log data set [13] is used for the experiments. The data set is separated and pre-processed in the same way as in [12]. Additionally, we exclude queries that are longer than 99 characters (<0.1% in the data set) to avoid bias in learning. In the background data, we remove queries that appear less than three times in order to remove noisy queries. The final background data contain 6,773,535 queries, and we sample the training (for LambdaMART), validation, and test data as in [12]. The training, validation, and test data contain 75,198, 30,993, and 32,044 prefixes, respectively, and each of the prefix is paired with at most ten query candidates. We compute MRR and PMRR from the top ten query candidates for each prefix as in [15].

5 RESULT ANALYSIS

The overall evaluation result is shown in Table 2. The first research question we address is "can language modeling approach perform as good as MPC for previously seen queries?" Interestingly, our generative language model NQLM(L)+WE performs similarly well on previously seen queries compared to the "wisdom of the crowd"-based approach, MPC, in MRR. When the candidates of NQLM(L)+WE are appended to candidates of MPC (NQLM(L)+WE+MPC), the model significantly outperforms MPC in MRR by providing extra candidates when MPC have less than ten candidates. NQLM(L)+WE even significantly outperforms MPC in PMRR. This means that our neural language model suggests next few words very well. This is especially useful in mobile search where the screen size is limited.

The next research question is "how well can our proposed model predict queries that are previously unseen?" While MPC completely fails to suggest queries that were not seen in the past, Mitra and Craswell's approaches [12] suggest some useful queries especially when more synthetic queries are used. Our generative approach

[1]The partial-matching candidate can be detected by a Python statement "`candidate==query or query.startswith(candidate+' ')`" assuming all candidates start with the prefix.

[2]The number of hidden units 1,532 is comparable to the number 1,500 in [17].

Table 2: MRR and PMRR evaluation results. The highest score for each metric is in bold face.

	MRR			PMRR		
Model	Seen	Unseen	All	Seen	Unseen	All
MPC [1]	0.428	0.000	0.171	0.566	0.000	0.225
Char. n-gram (n=7)	0.363	0.236	0.287	0.550	0.376	0.445
Mitra10K+MPC+λMART [12]	0.427	0.179	0.278	0.586	0.297	0.412
Mitra100K+MPC+λMART [12]	0.428	0.212	0.298	0.588	0.368	0.455
Proposed models						
NQLM(S)	0.381	0.287	0.325	0.557	0.460	0.499
NQLM(S)+WE	0.406	0.286	0.334	0.582	0.445	0.500
NQLM(L)+WE	0.419	0.303	0.349	0.589	0.465	0.514
NQLM(S)+MPC	0.433	0.287	0.346	0.580	0.460	0.508
NQLM(S)+WE+MPC	**0.434**	0.286	0.345	0.580	0.445	0.499
NQLM(L)+WE+MPC	**0.434**	0.303	**0.355**	0.580	0.465	0.511
NQLM(S)+MPC+λMART	0.428	0.288	0.344	**0.594**	0.465	0.516
NQLM(S)+WE+MPC+λMART	0.428	0.288	0.344	0.590	0.454	0.508
NQLM(L)+WE+MPC+λMART	0.428	**0.305**	0.354	0.593	**0.475**	**0.522**

NQLM(L)+WE significantly improves their best method by 42.9% in MRR and 26.4% in PMRR for unseen queries. When we leverage MPC and λMART (NQLM(L)+WE+MPC+λMART) as their approach does, the improvement becomes even bigger (43.9% in MRR and 29.1% in PMRR). We found that adding MPC and λMART does not consistently give us improvement in MRR over adding MPC only, which was also found in [12]. However, adding MPC+λMART outperforms adding MPC in PMRR. Meanwhile, the character-level n-gram language model (Char. n-gram) performs worse than Mitra and Craswell's methods for seen queries, but it outperforms them for unseen queries, surprisingly. However, with a better probability estimation, our neural language models significantly outperform Char. n-gram for both seen and unseen prefixes.

NQLM(S)+WE yields a significant improvement over NQLM(S) for seen queries, but it performs similarly or worse than NQLM(S) for unseen queries. Since the word embedded space character approach, NQLM(S)+WE, is supposed to work better than NQLM(S) when the words in prefix are interpretable (*i.e.*, previously seen), this result makes sense. This result is also consistent with that in [3] where character-only version works better than the one with word embedding, for a data set with a high Out-Of-Vocabulary rate. Meanwhile, a larger model NQLM(L)+WE adds extra significant improvement over a small one NQLM(S)+WE for both unseen and seen queries, with a more expressive power but a more expensive computation.

For a qualitative analysis, we show the top query candidates for a popular prefix in the Table 3.[3] Indeed, the candidates from NQLM(L)+WE are similar to those from MPC. One interesting thing to note is that the ranks of query candidates such as "www my", "www myspace", and "www yahoo" in NQLM(L)+WE are higher than their ranks in MPC. The related and more popular queries such as "www yahoo com" and "www myspace com" could have promoted those shorter versions since their generation probabilities are correlated. Such correlation may negatively affect the prediction of full queries, and this is the limitation of our generative model.

Table 3: Top query candidates from MPC and our generative model NQLM(L)+WE for a popular prefix "www".

MPC	NQLM(L)+WE
www google com	www google com
www yahoo com	www yahoo com
www myspace com	www myspace com
www google	www google
www ebay com	www hotmail com
www hotmail com	www my
www mapquest com	www myspace
www myspace	www mapquest com
www msn com	www yahoo
www bankofamerica com	www disney channel com

[3]Those for unseen prefixes by NQLM(L)+WE are shown in Table 1.

6 CONCLUSIONS AND FUTURE WORK

We proposed the first neural language model for QAC to handle unseen prefixes. We also provide an algorithm that generates query candidates from a prefix using a language model. Our empirical results show that (1) the neural language model performs as good as the "wisdom of the crowd" method (MPC) for previously seen queries, (2) it improves the state-of-the-art QAC method by 43% in MRR for previously unseen queries, and (3) the proposed word embedding technique yields a significant improvement for seen queries.

Our model was not evaluated on multiple data sets, and this is the limitation of our experiments. Our work can be further extended in several directions. Our neural language model can be easily adapted to personalized QAC [15] by remembering a specific user's query session state or a specific gender's general state. Modeling time-sensitive QAC [16] is also an interesting direction of neural language models. These directions are left as our future work.

REFERENCES

[1] Ziv Bar-Yossef and Naama Kraus. 2011. Context-sensitive query auto-completion. In *Proceedings of the 20th international conference on World wide web*. ACM, 107–116.

[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research* 3, Feb (2003), 1137–1155.

[3] Piotr Bojanowski, Armand Joulin, and Tomas Mikolov. 2015. Alternative structures for character-level RNNs. *arXiv preprint arXiv:1511.06303* (2015).

[4] Jan A Botha and Phil Blunsom. 2014. Compositional Morphology for Word Representations and Language Modelling.. In *ICML*. 1899–1907.

[5] Fei Cai, Maarten de Rijke, and others. 2016. A survey of query auto completion in information retrieval. *Foundations and Trends® in Information Retrieval* 10, 4 (2016), 273–363.

[6] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (2012).

[7] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[8] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2015. Character-aware neural language models. *arXiv preprint arXiv:1508.06615* (2015).

[9] Matti Lankinen, Hannes Heikinheimo, Pyry Takala, Tapani Raiko, and Juha Karhunen. 2016. A Character-Word Compositional Neural Language Model for Finnish. *arXiv preprint arXiv:1612.03266* (2016).

[10] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model.. In *Interspeech*, Vol. 2. 3.

[11] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE, 5528–5531.

[12] Bhaskar Mitra and Nick Craswell. 2015. Query auto-completion for rare prefixes. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 1755–1758.

[13] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. 2006. A picture of search.. In *InfoScale*, Vol. 152. 1.

[14] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 373–374.

[15] Milad Shokouhi. 2013. Learning to personalize query auto-completion. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 103–112.

[16] Milad Shokouhi and Kira Radinsky. 2012. Time-sensitive query auto-completion. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 601–610.

[17] Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 1017–1024.

[18] Idan Szpektor, Aristides Gionis, and Yoelle Maarek. 2011. Improving recommendation for long-tail queries via templates. In *Proceedings of the 20th international conference on World wide web*. ACM, 47–56.

[19] Saúl Vargas, Roi Blanco, and Peter Mika. 2016. Term-by-term query auto-completion for mobile search. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*. ACM, 143–152.

[20] Qiang Wu, Christopher JC Burges, Krysta M Svore, and Jianfeng Gao. 2010. Adapting boosting for information retrieval measures. *Information Retrieval* 13, 3 (2010), 254–270.