DeepIntent: Learning Attentions for Online Advertising with Recurrent Neural Networks

Shuangfei Zhai[*]
Dept. of Computer Science
Binghamton University
Binghamton, NY, USA
szhai2@binghamton.edu

Keng-hao Chang
Microsoft
Sunnyvale, CA, USA
kenchan@microsoft.com

Ruofei Zhang
Microsoft
Sunnyvale, CA, USA
bzhang@microsoft.com

Zhongfei (Mark) Zhang
Dept. of Computer Science
Binghamton University
Binghamton, NY, USA
zhongfei@cs.binghamton.edu

ABSTRACT

In this paper, we investigate the use of recurrent neural networks (RNNs) in the context of search-based online advertising. We use RNNs to map both queries and ads to real valued vectors, with which the relevance of a given (query, ad) pair can be easily computed. On top of the RNN, we propose a novel attention network, which learns to assign attention scores to different word locations according to their intent importance (hence the name DeepIntent). The vector output of a sequence is thus computed by a weighted sum of the hidden states of the RNN at each word according their attention scores. We perform end-to-end training of both the RNN and attention network under the guidance of user click logs, which are sampled from a commercial search engine. We show that in most cases the attention network improves the quality of learned vector representations, evaluated by AUC on a manually labeled dataset. Moreover, we highlight the effectiveness of the learned attention scores from two aspects: query rewriting and a modified BM25 metric. We show that using the learned attention scores, one is able to produce sub-queries that are of better qualities than those of the state-of-the-art methods. Also, by modifying the term frequency with the attention scores in a standard BM25 formula, one is able to improve its performance evaluated by AUC.

Keywords

Deep Learning; RNN; Attention Mechanism; Online Advertising; Sponsored Search; Query Rewriting; BM25

[*]This work was completed during the first author's internship at Microsoft.

KDD '16, August 13-17, 2016, San Francisco, CA, USA

© 2016 ACM. ISBN 978-1-4503-4232-2/16/08…$15.00

DOI: http://dx.doi.org/10.1145/2939672.2939759

1. INTRODUCTION

In the domain of sponsored search, it is essential to understand the search intent from queries as well as the selling intent from ads. Precise representation for both the queries and the ads is the foundation for quality retrieval and a stepping stone for higher user engagement. Succinct representations for queries is also important as it allows more ads to be selected, implying denser bidding landscape and stronger marketplace horsepower. For example, for the query "surface pro 3 keyboard", ads retrieved for shortened sub-queries "surface pro 3" and "surface 3" are not desirable as they do not precisely capture the user intent, which is in fact a keyboard instead of a tablet. On the other hand, if a user queries "Microsoft office for Mac", ignoring the term "Microsoft" does not cause any intent shift and enables one to retrieve more related ads. Representing ads is equally important for the success of a sponsored ads search engine due to a similar reasoning.

In this paper, we adopt recurrent neural networks (RNNs) as the building block to learn desired representations from massive user click logs. RNNs fall into the family of deep learning models which specifically focus on learning multiple levels of representations. RNNs are very suitable for modeling sequential data (such as natural language), and have recently achieved remarkable performance on tasks such as machine translation [22, 1] and image annotation [24]. In this paper, we propose a novel attention network which is stacked on top of an RNN and learns to assign attention scores to words within a sequence (either a query or an ad). Methodology-wise, we interpret attention as a pooling method, similar to mean pooling and max pooling, which compresses a sequence of vectors to a single vector. In contrast to mean pooling and max pooling however, attention based pooling is adaptive, which is able to assign large weights to "important" words and vice versa. Compared with using the last state of an RNN as the output (which we denote as last pooling in this paper), attention based pooling has no difficulty modeling long sequences, as it considers different word locations in an even manner. We consider several types of the underlying RNN, varying from both the one directional and bidirectional versions of vanilla RNN and LSTM [7]. We train our model with a large amount of click logs sampled from a commercial search engine, in the same fashion as [9]. Specifically, we push the model to learn vector representations for both queries and ads such that the click behavior can be well predicted.

We conduct two types of evaluations. The first one is directly taking the output of the RNN as the vector representation of queries and ads. We show that the attention layer improves the quality of learned vector representations by a vanilla RNN. On LSTM, the vectors learned by attention based pooling match those by last pooling, but with the additional advantage of much better model interpretability (which is directly used in the subsequent tasks). In the second set of evaluation, we investigate the quality of the learned attention scores. In particular, we perform query rewriting by selecting the sub-queries corresponding to terms that are assigned high attention. We show that we are able to outperform two existing query rewriting methods by a large margin. We further propose a novel way of utilizing the attention scores to enhance the term space representations, by modifying the term frequency part of the standard BM25 metric with the attention scores. We show that the learned attention scores are able to improve the AUC of BM25, with bidirectional LSTM being the model yielding the largest improvement.

The contribution of the paper is summarized as follows:

- We introduce an attention and RNN based deep architecture for modeling queries and ads in online advertising, and we have verified that using the attention based pooling successfully helps RNNs learn better vector representations for queries and ads.

- We propose novel ways of utilizing the attention scores learned from the deep model. We show that our learned query attention scores can be successfully applied to a query rewriting task, yielding quality rewrites that outperform existing methods. We also propose to use the learned ads attention scores to modify the term frequency of the standard BM25 metric, and demonstrate improved performance.

2. MODEL

As shown in Figure 1, DeepIntent consists of two parts: 1) a query and an ad encoder that take a word sequence as input and outputs a real-valued vector 2) a loss function that provides the signal to guide the learning of the two encoders. The encoders for both the query and ads share the same architecture, which is shown in Figure 2. We will first introduce each component of the encoder in detail, then discuss the construction of the loss function.

2.1 Input Layer

The input to an encoder is a word sequence $\mathbf{x} = \{x_1, ..., x_T\}$ where $x_t \in R^V$ is a one-hot-vector representation of the t-th word and V is the vocabulary size. T is the length of the sequence, which varies for different sequences. Note the difference from the traditionally used BoW representations, as we strictly consider the sequential order.

2.2 Word Embedding Layer

The second layer is a word embedding layer. For each time step t, the word embedding layer transforms the one-hot-vector x_t into a low-dimensional dense vector e_t via a linear mapping

$$e_t = W_{emb} x_t, \ s.t. \ W_{emb} \in R^{d_{emb} \times V}, \tag{1}$$

where each column of W_{emb} corresponds to a word in the vocabulary. The word embedding layer reduces the dimensionality of the input from the vocabulary size which could be up to millions to d_{emb} which is usually several hundreds, thus regularizing the model size.

2.3 RNN Layer

The output of the word embedding layer $\{e_1, ..., e_T\}$ is then fed into an RNN layer. An RNN updates a hidden state $h_t \in R^{d_h}$ given e_t at the t-th time step and the previous hidden state h_{t-1} in a recurrent formula:

$$h_t = \mathcal{H}(e_t, h_{t-1}), \tag{2}$$

where $\mathcal{H}$ is a nonlinear transformation of choice. The simplest form of an RNN is:

$$h_t = \sigma(W_{eh} e_t + W_{hh} h_{t-1} + b_h), \tag{3}$$

where $W_{eh} \in R^{d_h \times d_{emb}}$, $W_{hh} \in R^{d_h \times d_h}$, $b_h \in R^{d_h}$ are the parameters to be learned; $\sigma(\cdot)$ is a nonlinear activation function, eg., ReLU $max(0, x)$. In this case, $\mathcal{H}$ takes a fairly simple form consisting of a linear mapping followed by a simple nonlinear function. There are also more sophisticated forms of $\mathcal{H}$ available. For example, LSTM [7] has recently attracted much attention due to its ability to model long term dependency in sequences, which updates its hidden state as well as a cell state as follows:

$$\begin{aligned} i_t =& \sigma(W_{ei} e_t + W_{hi} h_{t-1} + b_i) \\ f_t =& \sigma(W_{ef} e_t + W_{hf} h_{t-1} + b_f) \\ c_t =& f_t \cdot c_{t-1} + i_t \cdot tanh(W_{ec} e_t + W_{hc} h_{t-1} + b_c) \\ o_t =& \sigma(W_{eo} e_t + W_{ho} h_{t-1} + b_o) \\ h_t =& o_t \cdot tanh(c_t), \end{aligned} \tag{4}$$

where $\cdot$ denotes the element-wise product between vectors; $tanh$ is the hyperbolic tangent function. LSTM works by maintaining a cell vector c_t in addition to the hidden state h_t, and introduces input gate i_t, forget gate f_t, output gate o_t to control the update dynamics of the cell and hidden units. Detailed discussions of the advantages of LSTM can be found in [7, 6], which is omitted in this paper due to the space limit.

Besides, it is also useful to build two RNNs taking a sequence in the forward and backward directions, respectively. This results in a bidirectional recurrent neural network (BRNN):

$$\begin{aligned} &\overrightarrow{h}_t = \overrightarrow{\mathcal{H}}(e_t, h_{t-1}), \ \overleftarrow{h}_t = \overleftarrow{\mathcal{H}}(e_t, h_{t+1}) \\ &h_t = concatenate(\overrightarrow{h}_t, \overleftarrow{h}_t), \end{aligned} \tag{5}$$

where the hidden state h_t is achieved by concatenating the corresponding hidden states from the forward and backward passes. BRNN breaks the asymmetry along the time steps and leverages the context around each step in a more balanced way.

Now each h_t can be considered as a representation of the t-th word. Due to the recurrent update of hidden states, h_t also takes the whole sequence into consideration. Hence, h_t is conceptually able to better characterize the semantics of the t-th word location than e_t, which is unaware of the local context. This is especially important for words that

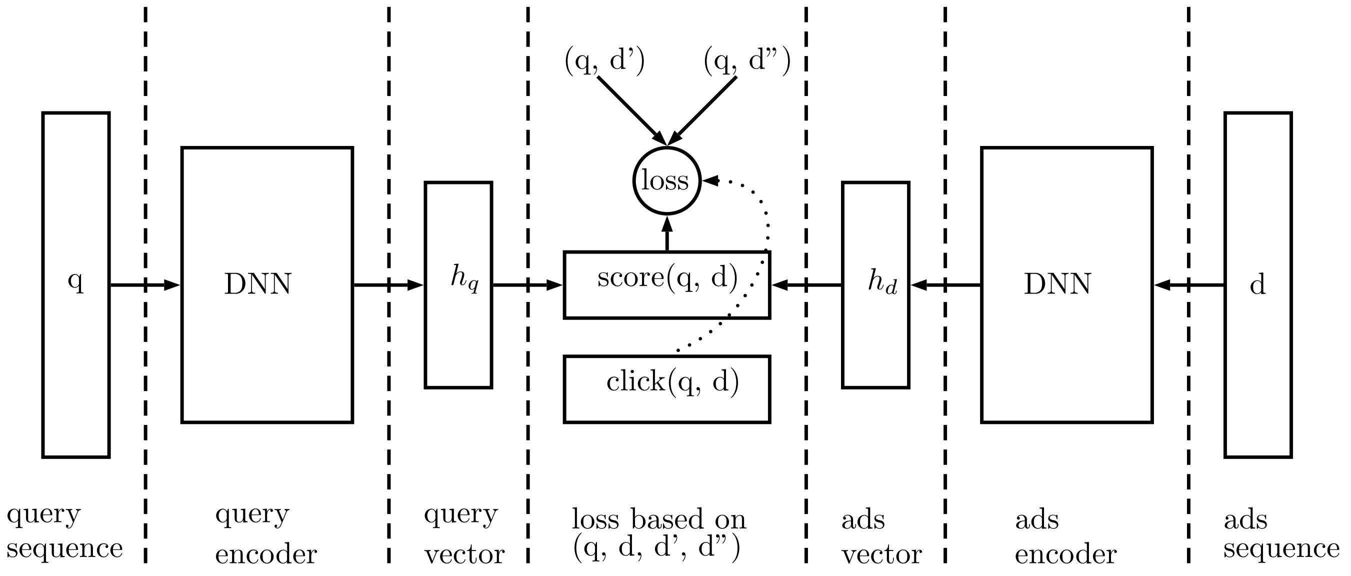

Figure 1: The architecture of the model, which consists of a query and an ad encoder, followed by a loss guided by the click.

have multiple senses, where only with the exact context one is able to tell the meaning of a particular occurrence.

2.4 Attention Based Pooling

The output of an RNN is another sequence of vectors $\mathbf{h} = \{h_1, ..., h_T\}$ with $h_t \in R^{d_h}$. It is thus desirable to further compress the sequence to a single vector to make it easy to build a loss function to facilitate training. In this paper, we denote this step as *pooling*.

The most straightforward approach is to take the last hidden state h_T as the representation for the sequence (for BRNN, this corresponds to *concatenate*$(\overrightarrow{h}_T, \overleftarrow{h}_1)$). In order to make this comparable to other pooling methods, we denote this approach as last pooling which is a slightly unorthodox notation. Note that as in Equation 2, h_T is a function of the whole sequence $\mathbf{x}$; thus theoretically h_T should be able to capture all the information contained in $\mathbf{x}$. However, this simple approach causes the long term dependency problem where the last state is forced to remember inputs that are many steps away. LSTM suffers less from this problem, thanks to its sophisticated update scheme with the help of cell and gate units.

Mean pooling is another popular choice which takes the mean of the hidden states from all time steps: $h = \frac{1}{T}\sum_{t=1}^{T} h_t$. It is obvious that long term dependency is no longer the main issue as every time step is taken into account in the first place. This is especially intuitive when considering a BRNN as conceptually every word contributes equally to the final vector representation, without any bias towards the last (or first) few words. However, this property is also a drawback of mean pooling, as it is unable to distinguish words that dominate the intent of a query or an ad from those that do not.

Max pooling is similar to mean pooling where instead of taking the average it takes the element-wise max of all the hidden states: $h = \max(h_1, ..., h_T)$. The max operation also provides an additional level of nonlinearity, making it a nonlinear transformation. As a result, max pooling typically works better and is arguably more widely used than mean pooling in practice. Despite this, max pooling suffers from the same problem as mean pooling being unable to perform proper credit assignment for individual words.

Motivated by the drawbacks of the above mentioned methods, we propose a novel attention based pooling, which constitutes the main contribution of this paper. We adopt the idea from the sequence to sequence learning machine translation literature [1], but extend it to a more general setting that makes it applicable to cases whenever a global pooling is needed. To be concrete, attention based pooling represents a sequence by a weighted sum of the vector representations of all time steps. In particular, the weights are adaptive to the content of each time step, which makes it able to perform proper credit assignment to time steps according to their importance. Mathematically, it takes the form as follows:

$$h = \sum_{t=1}^{T} a_t h_t, \; a_t = \frac{\exp(s(h_t; \theta))}{\sum_{t=1}^{T} \exp(s(h_t; \theta))}. \tag{6}$$

Here $s(\cdot; \theta)$ (which we call the attention network) is a function that maps a vector h_t to a real valued score. a_t, which we call *attention*, is the normalized score across the whole sequence, which indicates the importance of the corresponding time step t. Compared with mean pooling, Equation 6 replaces the static weight for each time step $\frac{1}{T}$ with a_t, which is adapted to the content of step t. This allows the model to pay higher attention to important words and vice versa. There are two key ingredients that make this possible. First is that h_t should form a precise representation of the t-th word with the context in consideration, and this is achieved by the underlying RNN. Second is that $s(\cdot; \theta)$ must have enough capacity to be able to distinguish between important and unimportant words. For example, a simple linear function would not be able to do so due to the large volume of the semantic space spanned by queries and ads. As a result, we implement $s(\cdot; \theta)$ with yet another neural net-

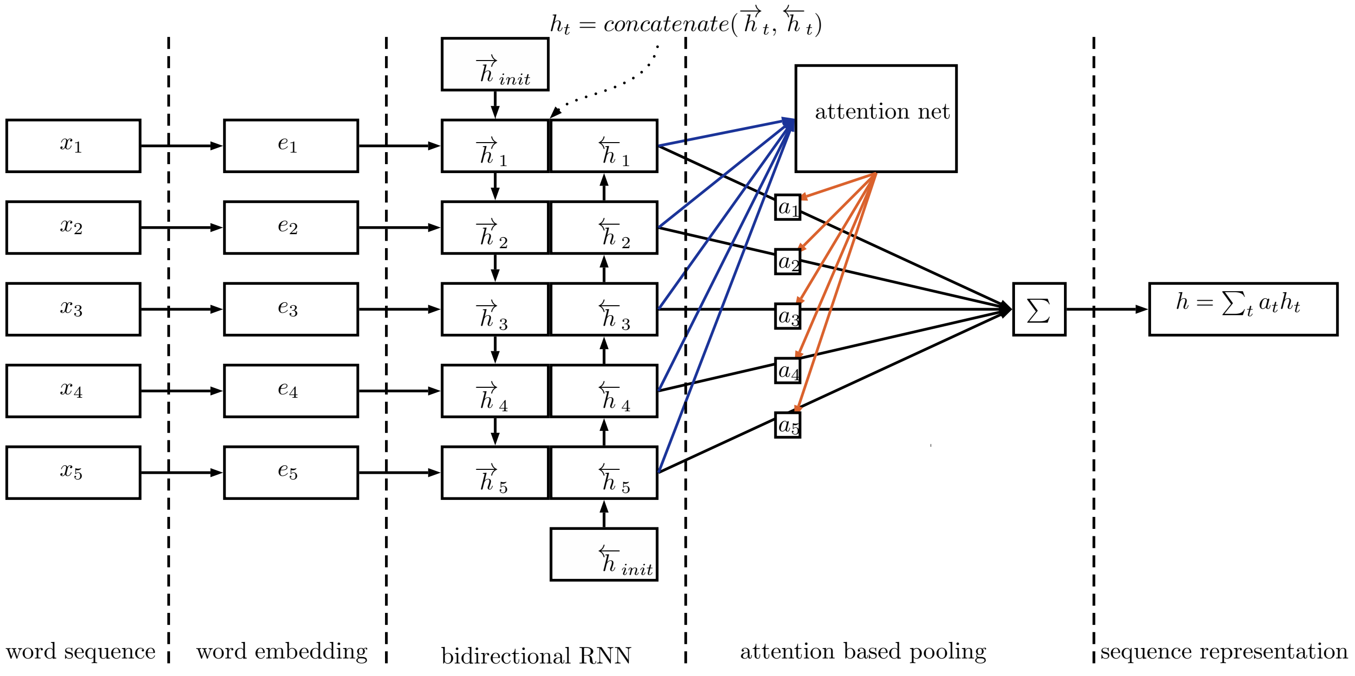

Figure 2: The encoding architecture of attention based pooling, demonstrated with a bidirectional RNN.

work, whose parameters are learned jointly with those of the underlying RNN (hence the name attention network).

The attention scores give us a straightforward way for visualizing a deep neural network based model. This greatly enhances the interpretability of a deep model and allows one to perform deeper analysis of its behavior. In addition, one is also able to extract the words with high attention scores as the dominant intent of a sequence, which can be later fed to existing keyword based information retrieval systems. As the attention scores are computed with a deep architecture which directly models local context, they provide more precise and informative weightings of the words in a sequence. In the experiments, we conduct a series of qualitative and quantitative evaluations and demonstrate that we are able to learn meaningful and useful attention scores.

2.5 Loss Function

As shown in Figure 1, after obtaining the vector representations from query and ad encoders defined previously, we need to define a proper loss function to make learning possible. We first observe that unsupervised learning is unsuitable as it is difficult to make the distinction between important and unimportant contents without supervision. Luckily, for commercial search engines, large amounts of user click logs are available, which provide valuable information about the relevance of a given query and ad pair. Intuitively, the more clicks a pair has, the better a match it is. To this end, we adopt a loss function that is similar to that in [9] which tries to maximize the similarity of the query and ad pairs that have large click counts. Formally, let (q, d^+) be a clicked (query, ad) pair, and $\{d_1^-, ..., d_n^-\}$ be a set of ads that have zero click count with q. We denote $h_q(\cdot)$ and $h_d(\cdot)$ as the encoding functions according to Equation 6 for query and ad sequences, respectively. We then formulate our loss function as follows:

$$J(\theta) = -\sum_{(q,d^+)} \log \frac{\exp(score(q,d^+))}{\exp(score(q,d^+)) + \sum_{i=1}^{n} \exp(score(q,d_i^-))}$$
$$s.t.\ score(q,d) = h_q(q)^T h_d(d), \tag{7}$$

where we have slighted overloaded the notation to let $\theta = \{W_{q,emb}, \theta_{q,rnn}, \theta_{q,attention}, W_{d,emb}, \theta_{d,rnn}, \theta_{d,attention}\}$ be the union of the parameters for both the query and ad encoder. In this way, all the parameters defined previously are learned with the goal to correctly match related (query, ad) pairs. Note that in Equation 7, the summation is denoted as summing over only (q, d^+), where $\{d_1^-, ..., d_n^-\}$ are not explicitly expressed. This is to highlight our implementation where for each (q, d^+), we randomly sample n ads $\{d_1^-, ..., d_n^-\}$ that are not clicked by q; we then form a pseudo $n + 1$ way classification problem where d^+ is considered the positive class, and all the rest are negative. The probability is computed according to the inner product of the vector representation of the query with all of those ads. The adoption of negative sampling greatly reduces the cost of computing the loss, which makes it scale linearly with the number of clicks. In this way, we can directly apply maximum likelihood estimation and minimize $J(\theta)$ with standard gradient descent, and train our model on a large set of click logs.

3. EXPERIMENTS

3.1 Training Dataset

We use user click logs sampled from a commercial "product ads" search engine as training data, where only queries with clear product intent are selected. A total of about 15 million clicks are sampled from a month long of click logs,

	size	vocabulary	average length	clicks
query	6.4M	68K	4.1	15M
ads	5.1M	114K	9.3	15M

Table 1: The statistics of the training set.

which ends up with 6.4 million distinct user queries and 5.1 million distinct ads. We tokenize and represent each query (and ad) as a sequence of uni-gram terms. This yields a dictionary size of around 68 thousand and 114 thousand for queries and ads, respectively. The average length of queries and ads are 4.1 and 9.3 words, respectively. We summarize the statistics of the training dataset in Table 1.

3.2 Implementation Details

We use Theano [2] for the implementation and experiments are run on a Tesla K20 GPU. For all the models, we implement the attention network as a fully connected neural network with one hidden layer. $\sigma(\cdot)$ is set as ReLU for all the cases except for LSTM, where sigmoid is used. We train all the models with mini-batch gradient descent with Adadelta [25]. We use a held out validation set to monitor the training progress, and all the models are trained till the validation loss stops decreasing.

3.3 Evaluation of Vector Representations

We train a variety of models with different architectures to highlight the effectiveness of each component of our model. We summarize the description of all the models we have compared in Table 2. Here we choose the underlying encoding methods from the set {BoW, RNN, BRNN, LSTM, BLSTM}; and pooling methods from the set {max pooling, last pooling, attention based pooling} when applicable. For BoW, the word order is ignored and the corresponding h_t is computed simply as $h_t = \sigma(Wx_t + b)$. And a proper pooling method is then applied to $\{h_1, ..., h_T\}$ as for RNNs. Note that it does not make sense to apply last pooling for BoW, so it is ignored in our experiments. For LSTM and BLSTM, max pooling is ignored as it does not give competitive performance to last pooling in practice. For all the combinations, the dimension of the final vector representation is fixed as 400; the activation function σ is set as ReLU for all the models except for LSTM where we use the sigmoid function. All of them are trained with the same loss function defined in Section 2.5.

As a quantitative evaluation, we test all the models on a fully annotated test set. The test set contains around 966 thousand (query, ad) pairs where each pair is labeled by a group of trained human judges according to the relevance between the query and the ad. Each label ranges in {bad, fair, good, excellent}. The pairs are sampled from the early selection stage of a commercial ads search engine, where there are a significant amount of low quality selected ads to be pruned out in downstream processing. We use AUC (area-under-curve of the receiver operating characteristic plot) [20] as the metric for evaluation, by considering the bad label as the negative class and the rest labels as the positive class (fair, good and excellent). This results in a test set consisting of 318 thousand positive and 597 thousand negative pairs. We briefly summarize the statistics of the testset in Table 3. For each model, we use the cosine similarity $\frac{<h_q(q), h_d(d)>}{\sqrt{\|h_q(q)\|_2^2}\sqrt{\|h_d(d)\|_2^2}}$ instead of the inner product as the relevance score for a pair (q, d). The AUC is then computed according to the scores for all the pairs in the test set. Our results are presented in Figure 3.

The first thing we observe from Figure 3 is that all the models that consider word order significantly outperform BoW, regardless of the pooling method being used. Given that all the models output vectors with the same dimensionality, this demonstrates the benefit of considering word order. Moreover, bidirectional models tend to work comparable to or better than the corresponding one directional variants. In particular, we observe an improvement of BRNN over RNN with all the three pooling methods. BLSTM on the other hand achieves comparable performance compared with LSTM. Note that with the same output dimensionalty, bidirectional models have less parameters compared with their one directional counterpart. To see this, consider an RNN with input dimension d_{in} and hidden size d_h, the number of total parameters are $d_{in} \times d_h + d_h \times d_h$, ignoring the bias term. The number of parameters for the corresponding BRNN is $2 \times d_{in} \times \frac{d_h}{2} + 2 \times \frac{d_h}{2} \times \frac{d_h}{2} = d_{in} \times d_h + \frac{1}{2} d_h \times d_h$. The comparison between LSTM and BLSTM can be seen similarly.

When comparing different pooling methods, we see that for BoW, RNN and BRNN, attention based pooling significantly outperforms max pooling and last pooling. For LSTM, it is interesting to see that attention based pooling is slightly worse than last pooling; on BLSTM, attention based pooling matches last pooling. Also, LSTM and BLSTM achieve the best results evaluated by the vector representations, which makes LSTM the most effective encoder.

In order to gain further insight of the behavior of attention based pooling, we have visualized a sample query and the top five matched ads by a BRNN in Figure 4. The query is chosen from the validation set, which is not seen by the model during training. The height of the bar indicates the value of the attention scores. From Figure 4, we see that all returned ads are of good quality which correctly match the intent of the query. In particular, the attention net correctly assigns larger weights to words that are interesting/important. This suggests that, our BRNN model is able to learn meaningful representations for words at each time; on the other hand, the attention net is able to perform correct credit assignment to different words according to their local contents. As a comparison, we have also visualized the attention scores assigned by LSTM and BLSTM on three ads in Figure 5. Interestingly, we see that for LSTM, the attention scores are obviously skewed to the right, which essentially puts most of the weights on the words that are close to the end of a sequence. Note that in the extreme case if all the attention is assigned on the last word, attention based pooling is equivalent to last pooling. This explains our observation that attention based pooling fails to improve upon last pooling on LSTM. For BLSTM, on the other hand, the attention net is able to perform correct credit assignment, highlighting words mostly related to the product type or brand. This consolidates our speculation about the benefit of using a bidirectional model to break the bias towards the later states of a recurrent neural network.

3.4 Query Intent Extraction with Attention

In the next set of experiments, we focus on the quality of the attention scores for queries. In particular, we evaluate its effect in query intent extraction systems. As we pointed

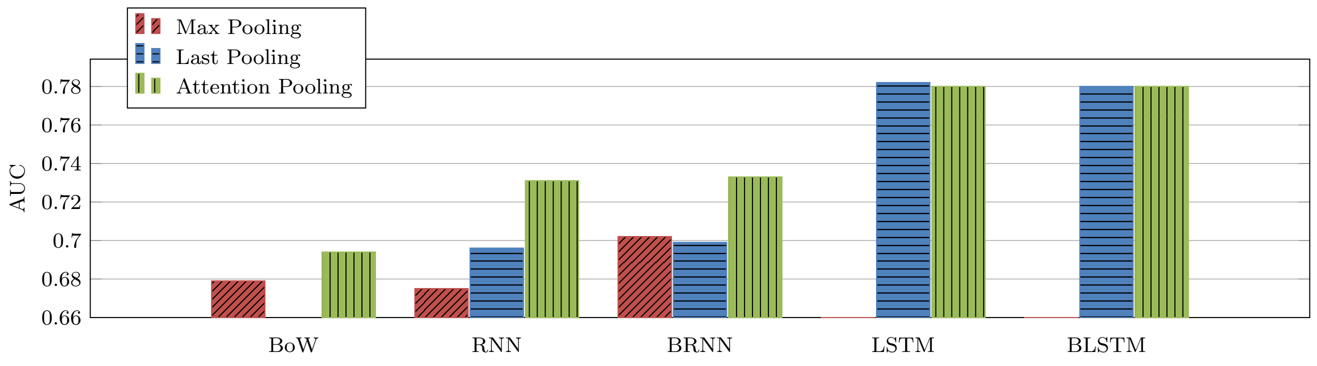

Figure 3: AUC evaluated on the test set of different models.

Model	Word Order	Max Pooling	Last Pooling	Attention Pooling
BoW	None	Yes	No	Yes
RNN	Forward	Yes	Yes	Yes
BRNN	Forward and Backward	Yes	Yes	Yes
LSTM	Forward	No	Yes	Yes
BLSTM	Forward and Backward	No	Yes	Yes

Table 2: The comparison of all the models.

# queries	# ads	# pairs	# positive	# negative
23K	915K	966K	318K	597K

Table 3: Statistics of the testset.

out earlier, the attention layer can be used to determine a subset of query words, which can then be used as query rewrites to increase ads selection density.

We compare the quality of rewrites chosen by attention layer against two baseline algorithms, by asking human judges to evaluate the relevance of the sub-queries. We choose two existing algorithms, namely PartialDrop (with an idea similar to [18]) and SmartIntent [15]. Both baseline algorithms use click information to identify term importance to generate sub-queries as rewrites. PartialDrop considers a term to be important and representing the query intent if dropping it leads to significant reduction of click-through rate (CTR). Removing such terms from a query thus are likely to cause intent shift. PartialDrop then works by choosing to remove certain terms while maintaining the CTR. SmartIntent, on the other hand, leverages query-ad click data by training a machine learning model to rank sub-queries. With the assumption that the distribution of ad bidded keywords is similar to that of queries, a model learned to rank keywords using CTR can then be applied to ranking sub-queries. SmartIntent first parses the query-ad bipartite click graph to summarize the CTR contribution of each pair of terms on both ends: (query term, ad bidded-keyword term). Given a pair of a query and a bidded keyword (or a sub-query), it then represents the relationship using mutual information of seen term pairs and graph walk [4] as features, and train a logistic regression on clicks as the prediction model.

As the test set for this task, we sample 1000 queries with product intent, and apply PartialDrop, SmartIntent on it, respectively. Note that as one shortens a query, the information carried in a sub-query monotonically decreases as the number of words remaining decreases. As a result, we use the following experiment protocol: for each sub-query that PartialDrop or SmartIntent generates, we generate an attention-based counterpart: a sub-query with the same number of words as the baseline, selected according to the assigned attention scores. That is, if a baseline algorithm generates a sub-query of length K, we are interested in whether the attention net is able to choose another set of K query words with higher quality. For example, if a query is “awesome lego star wars sets”, a 4-word sub-query “lego star wars sets” is considered more relevant compared to “awesome star wars sets”. For this experiment, we use the attention scores from a BRNN trained on the dataset described in Section 3.1.

As a quantitative evaluation, we submit all the (query, rewrite) pairs produced by different algorithms to human judges to label the query-rewrite quality. Each time a judge is presented with a pair of a query and a rewrite (a sub-query in our case), the judge needs to rate whether the rewrite is relevant to the query, with rating label ranges in {same, superset, subset, overlap, disjoint}. This rating system is designed to analyze whether the “product intent space” of a query is intact compared to that of a rewrite. If the product space of a query has the “same” or is a “superset” of that of a rewrite, the rewrite is considered of good quality. On the other hand, if the query product space is a “subset” of the rewrite product space, the query intent is shifting wider in the rewrite. The rewrite could select widened, poor results. The same concept applies to “overlap” and “disjoint”.

We summarize the comparison results in Table 4. We see that compared with both PartialDrop and SmartIntent, attention scores yield substantially larger portions of rewrites with good quality. We have also conducted paired t-tests on the labeling results, by considering “same” and “superset” labels as good quality rewrites (quality score = 1), and the rest as bad quality (quality score = 0), which indicates that attention-based rewrite has significant better qualities than both PartialDrop ($p = 0.002, t = 3.102, n = 877$, two-tailed) and SmartIntent ($p = 0.001, t = 3.302, n = 787$, two-tailed).

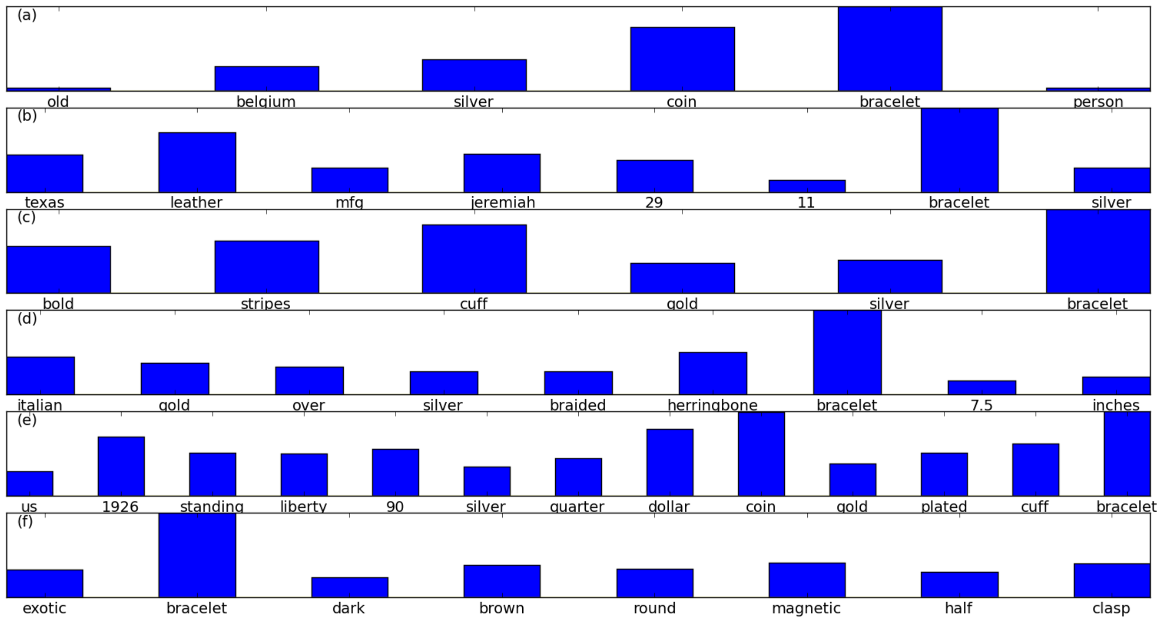

Figure 4: Attention score visualization of a query (a) and 5 top ranked ads (b)(c)(d)(e)(f) by BRNN.

method	# good	% good	method	# good	% good
attention	228	0.275	attention	212	0.286
PD	192	0.222	SI	170	0.225

Table 4: The comparison of our method (attention) with PartialDrop (PD) and SmartIntent (SI).

This strongly confirms that the attention network is able to perform correct credit assignment to words within a query. Also, the large improvement over the two baselines indicates the effectiveness of a deep model that directly models sequences over their shallow BoW based competitors.

3.5 Modified BM25 with Attention

In the next set of experiments, we proceed to evaluate the attention scores learned for ads. In particular, we propose a novel usage of the attention scores by incorporating it with the standard BM25 ranking function [21]. BM25 scores each document based on the query terms appearing in the document as follows:

$$score(q,d) = \sum_{i=1}^{n} IDF(q_i) \frac{TF(q_i,d)(k_1+1)}{TF(q_i,d)+k_1(1-b+b\frac{|d|}{avgdl})}, \tag{8}$$

where $TF(q_i,d)$ is term q_i's term frequency in document d, $|d|$ is the length of document d, and $avgdl$ is the average document length in corpus.

Attention scores can be used to calculate non-uniform, more representative term frequencies. Terms with higher importance should be rewarded with larger term frequency, and vice versa. As in Equation 9, we redistribute the term frequencies using the attention scores as weights, i.e. the component $a_j|d|$. In addition, we add a hyper-parameter λ to balance between the attention-based term frequency and the uniform term frequency (i.e. value = 1) of a document term. This "attention significance" parameter λ serves as a smoothing factor to avoid attention scores being too sharp to force certain term frequency to zero. A zero term frequency implies that matching such a term has no contribution to the relevancy between a query and a document, which is intuitively not desirable [16]. In addition, the optimal λ value (decided by validation set) indicates how useful the learned attention is to the BM25 application. If the λ is zero, it falls back to the uniform term frequencies. We then propose the modified BM25 by replacing the $TF(q_i,d)$ in Equation 8 with attention-based $TF'(q_i,d)$ as follows:

$$TF'(q_i,d) = \sum_{j:d_j=q_i} \lambda \cdot a_j|d| + (1-\lambda)\cdot 1 \tag{9}$$

We compute the AUC of the modified BM25 using the same labeled test set as in Section 3.3 (Table 3). We take 70% of the dataset as the validation set, on which grid search is applied in the range $\lambda = \{0, 0.1, 0.2, ..., 1\}$. The AUC is then computed with the selected optimal λ on the 30% remaining subset, with k_1 set to 2.0 and b set to 0.75 in Equation 8. We compare the attention scores learned by RNN, BRNN, LSTM and BLSTM. The test AUC numbers along with the optimal λ's for each model can be found in Figure 6.

The optimal λ values for RNN, BRNN, and BLSTM are equal or close to one, indicating the attention learned with RNN, BRNN, and BLSTM are good quality. In addition, BLSTM achieves the highest AUC gain, outperforming the standard BM25 by over one absolute percent. This is also

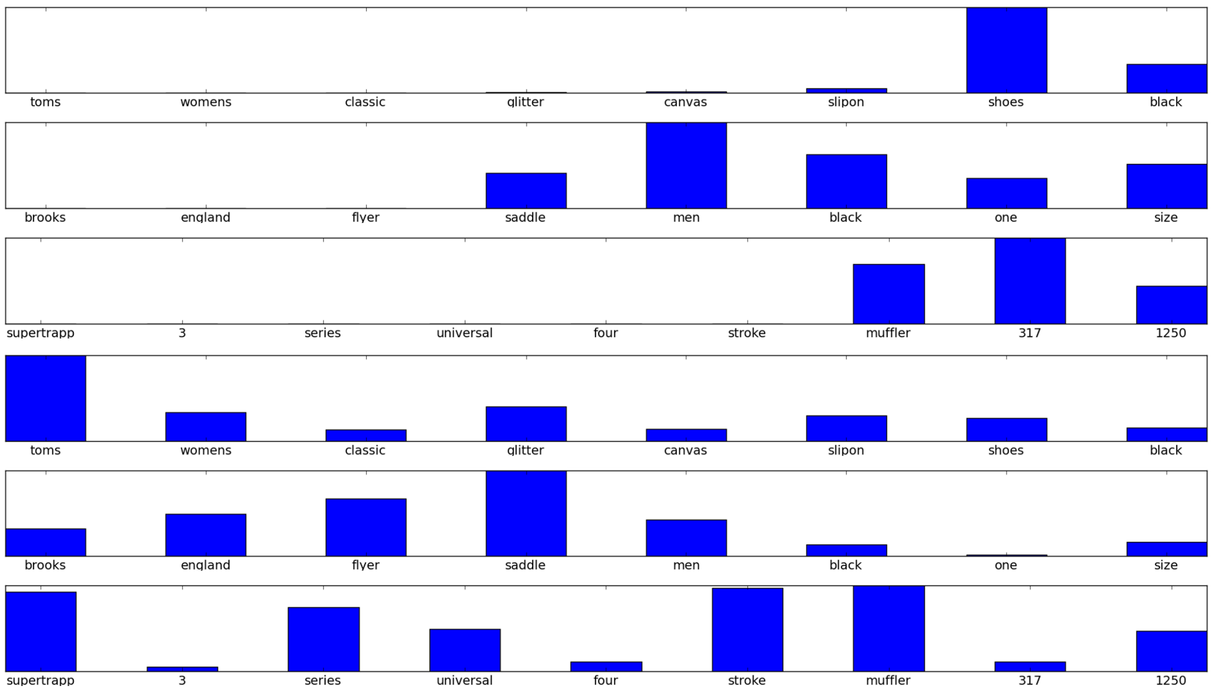

Figure 5: Visualization of the attention scores assigned on three ads. Top three: LSTM, bottom three: BLSTM.

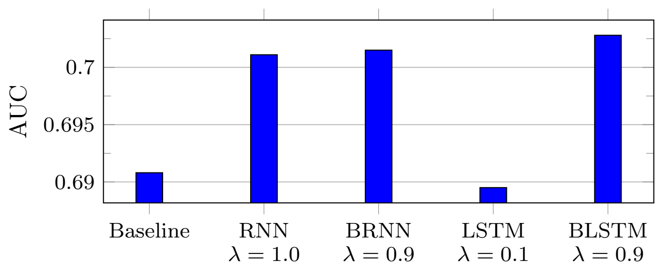

Figure 6: The test AUC of BM25 modified with attention under different architectures, with optimal attention significance factor λ grid-searched in training split

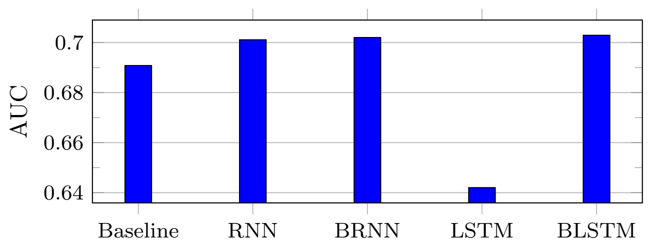

Figure 7: The test AUC of BM25 modified with attention under different architectures, with attention significance factor λ blindly set to 1.0

consistent with Figure 3 where BLSTM yields better vector representations than RNN and BRNN. On the contrary, the λ factor for LSTM is close to zero (which actually hurts the performance), implying the attention scores learned with LSTM is not useful. Figure 7 shows the test AUC where we blindly set λ to 1. Without any factor of smoothing, the attention learned with LSTM significantly degrade the AUC. This also correlates with our findings in Figure 3, where the attention based pooling does not provide improvement over last pooling.

Figure 5 gives an intuitive explanation of the behavior of the attention network on LSTM and BLSTM, where we see that the attention scores learned on LSTM commonly skew to the right. This is sensible because with the usage of gating units and cell units, LSTM is able to (and hence tend to) use the last hidden state to capture the information presented in the whole sequence. As a result, the effect of the attention network is compromised where it can lazily assign high attentions to the rightmost states, while incorrectly ignoring the ones further to the left. On the contrary, with BLSTM, the skewness problem does not exist, as shown in Figure 5. The attention scores are meaningful. For example, the important brand word ”toms” for the first shoes ad and ”superstrapp” for the third muffler ad are recognized, whereas with LSTM in Figure 5 they have close to 0 attention. Here the bidirectional recurrence helps recognize the intent that stands out in a position, without being biased by the history as in unidirectional recurrence. Plus with the quality control gates provided by LSTM, BLSTM attention modifies BM25 to achieve highest AUC compared to RNN and BRNN.

4. RELATED WORK

There has been a long thread of learning representations for information retrieval systems. Bag of words and its variants such as TF-IDF played a central role due to its simplicity and effectiveness. Vector space models (VSMs) [23] take a step further from BoW by mapping documents into a low dimensional vector space, which form more compact representations. Examples of VSMs include Latent Semantic Indexing [5], Probabilistic Latent Semantic Indexing [8] and Latent Dirichlet Allocation [3]. From one particular perspective, our work also falls into this category, as we are able to produce vector based representations for both queries and ads.

Recently, there has been a thrive to learn deep representations for natural language. The idea is that by using a deep architecture, one is able to learn highly nonlinear transformations of data, thus can exploit much more complicated patterns than simply using simple linear mappings. There has been successful approaches [10, 9, 19] demonstrating that deep architectures can significantly boost the performance of commercial search engines. Among these, [19] is closely related to our work, which corresponds to LSTM + Last pooling in our experiments. Our work further extends it by introducing the attention layer, which enables us to use the deep model to enhance term based tasks such as query rewriting and modified BM25. From this perspective, our work can be considered as enhancing the traditional term based IR systems with latent space deep models trained in an end-to-end fashion, which is to the best of our knowledge the first of its kind.

Methodology wise, our work extends the recently proposed attention mechanism [1, 14] from the NLP tasks such as machine translation and language modeling to the web search applications. Compared with [1], our attention module is static, that is able to directly compute the attention score without knowing the matching candidates. In a more general view, our attention based pooling is an extension of the pooling methods. As a result, it is also applicable in general wherever traditional pooling methods are needed.

There has been plenty query rewriting work for online advertising and information retrieval. Deleting query terms can help increase coverage and selection density. Substituting query terms or phrases solves the problem that a user search query may be an imperfect description of their information need. Both realm of work requires understanding the importance of a term or phrase in a query. Part of [12] simply considers query reformulation feedback by users. A term is considered not as important if a user removes the term from the previously issued query and resubmits. [13] uses point-wise mutual information to determine whether a phrase stands out comparing to adjacent terms. Using user click information [15] or pseudo relevance feedback [18] is also a good way to identify term importance. Intuitively speaking if dropping a term leads to significant reduction of click-through rate (CTR), or leads to reduced ability at discriminating relevant documents, such term is considered important. Both [15] and [18] leverages random walk algorithm [4] on a bipartite query-document graph to identify important query terms contributing CTR or relevance. Our work encompasses the essence of the prior work mentioned above. It leverages user clicks to guide the term importance so that a term with high attention (1) matches the intent of the whole query and (2) contributes to the clicks to relevant ads (and vice versa for the ad side attention). Instead of using bipartite graph, random walk algorithm, and some feature engineering, our attention learning is done through through DNN and embedding.

Okapi BM25 [21, 17] is a well known and widely used document ranking method in information retrieval. BM25 is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, through TF-IDF normalization. A known problem to BM25 is that many term weighting practices are insufficient in distinguishing informative words from uninformative ones. [11] addresses this problem by adjusting term frequencies in proportion to their semantic correlation to the categorical information of documents. Our work follows the same idea and takes a simplistic approach: we use attention to redistribute document length into term frequencies. This not only allows us to systematically evaluate attention scores, but also demonstrates the applicability of attention to TF-based information retrieval tasks.

5. CONCLUSIONS AND FUTURE WORK

We have proposed an attention based pooling on top of RNNs to model queries and ads in online advertising. The RNN module allows us to model word sequence, which is shown to be of great importance to accurately capture the meaning of a sequence. The attention based pooling module gives our model the ability to perform correct credit assignment to words within a sequence. We investigate various RNN architectures and conduct large scale experimental evaluation w.r.t. both the vector representations and the attention scores. In particular, we propose a novel way of applying the attention scores to query rewriting as well as a modified BM25 metric, and demonstrate the effectiveness of the learned attention for both queries and ads.

Possible extensions to our work also exist. For example, the fact that using the learned attention scores as term frequency yielding better BM25 metrics is actually a byproduct of our model. In other words, it is the result of optimizing a closely related objective. As a result, it is possible that we can formulate an objective function using the attention enhanced BM25 directly, while with the underlying encoder unchanged. We leave this as future work.

6. REFERENCES

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[4] K. Collins-Thompson and J. Callan. Query expansion using random walk models. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, pages 704–711, New York, NY, USA, 2005. ACM.

[5] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *JAsIs*, 41(6):391–407, 1990.

[6] K. Greff, R. K. Srivastava, J. Koutník, B. R.

Steunebrink, and J. Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015.

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

[8] T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999.

[9] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, pages 2333–2338. ACM, 2013.

[10] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 2333–2338, New York, NY, USA, 2013. ACM.

[11] R. Jin, J. Y. Chai, and L. Si. Learn to weight terms in information retrieval using category information. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 353–360, New York, NY, USA, 2005. ACM.

[12] R. Jones and D. C. Fain. Query word deletion prediction. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, pages 435–436, New York, NY, USA, 2003. ACM.

[13] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 387–396, New York, NY, USA, 2006. ACM.

[14] W. Ling, Y. Tsvetkov, S. Amir, R. Fermandez, C. Dyer, A. W. Black, I. Trancoso, and C.-C. Lin. Not all contexts are created equal: Better word representations with variable attention. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1367–1372, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[15] P. Liu, J. Azimi, and R. Zhang. Contextual query intent extraction for paid search selection. In *Proceedings of the 24th International Conference on World Wide Web Companion, WWW 2015, Florence, Italy, May 18-22, 2015 - Companion Volume*, pages 71–72, 2015.

[16] Y. Lv and C. Zhai. Lower-bounding term frequency normalization. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 7–16, New York, NY, USA, 2011. ACM.

[17] M. E. Maron and J. L. Kuhns. On relevance, probabilistic indexing and information retrieval. *J. ACM*, 7(3):216–244, July 1960.

[18] K. T. Maxwell and W. B. Croft. Compact query term selection using topically related text. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 583–592, New York, NY, USA, 2013. ACM.

[19] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. Ward. Deep sentence embedding using the long short term memory network: Analysis and application to information retrieval. *arXiv preprint arXiv:1502.06922*, 2015.

[20] F. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing induction algorithms. In *In Proceedings of the Fifteenth International Conference on Machine Learning*, pages 445–453. Morgan Kaufmann, 1997.

[21] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. pages 109–126, 1996.

[22] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[23] P. D. Turney, P. Pantel, et al. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37(1):141–188, 2010.

[24] K. Xu, J. Ba, R. Kiros, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.

[25] M. D. Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.