Learning to Reweight Terms with Distributed Representations

Guoqing Zheng
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
gzheng@cs.cmu.edu

Jamie Callan
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
callan@cs.cmu.edu

ABSTRACT

Term weighting is a fundamental problem in IR research and numerous weighting models have been proposed. Proper term weighting can greatly improve retrieval accuracies, which essentially involves two types of query understanding: interpreting the query and judging the relative contribution of the terms to the query. These two steps are often dealt with separately, and complicated yet not so effective weighting strategies are proposed. In this paper, we propose to address query interpretation and term weighting in a unified framework built upon *distributed representations* of words from recent advances in neural network language modeling. Specifically, we represent term and query as vectors in the same latent space, construct features for terms using their word vectors and learn a model to map the features onto the defined target term weights. The proposed method is simple yet effective. Experiments using four collections and two retrieval models demonstrates significantly higher retrieval accuracies than baseline models.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval models

General Terms

Algorithms, Experimentation

Keywords

Query term weighting, distributed representations, word vectors

1. INTRODUCTION

Performance of text search engines relies heavily on query understanding, of which one important problem is how to weight the contribution of each individual term to the retrieval score[1]. When proper weights (such as ground truth *term recall* weights [22]) are used, they can boost retrieval accuracies by up to 30% given relevance judgments. Properly setting the query term weights requires accurately interpreting and properly representing the query first. This is no easy task as query intent understanding itself is a diffcult problem in IR research [7].

In this paper, we attempt to address query interpretation and term weighing from a different angle, with a unified framework built upon recent advances in neural network language modeling [13, 3]. Recent research in the application of neural network to text problems exploits the co-occurrence of words to represent words by multidimensional vectors. Distributed representations learned from neural network based models [13, 3] are designed and shown to be effective for measuring semantic similarity among words and identifying similar neighbors for a given word. The mapping from words to vectors gives the ability to not only measure word-word similarity but also ways to represent the query in the same vector space from word vectors from its terms (such as taking the average of the word vectors of all terms as the vector representation for the query).

As proper query term weights reflects the relative importance of the term with respect to the query, specifically we propose to construct features from word vectors representations of terms and the query and to learn the relationship between the feature vectors and target term weight (such as the term recall weight [22] estimated from relevance judgments). A regularized linear regression problem from the feature vectors onto term weights is formulated and the predicted term weights are used for both bag-of-words queries and term dependency queries. We demonstrate the effectiveness of our method using two popular retrieval models, four standard test collections, word vectors developed from a variety of sources, and three baseline methods.

The contributions of our work are three fold. First, we join the work of distributed word vectors to the prediction of query term weights in IR, and propose a simple yet effective framework to predict effective term weights. Second, we observe significant improvement over the baseline mod-

[1]People use "term weight" to describe two different settings: one is the matching score a retrieval function assigns to a term and a document pair; the other is the relative importance of the term to the query. The first setting is handled by the retrieval model while the query model deals with the second one; in this paper, we refer "term weight" or "term reweighting" to that in the second setting.

SIGIR'15, August 09 - 13, 2015, Santiago, Chile.
© 2015 ACM. ISBN 978-1-4503-3621-5/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2766462.2767700.

els with two retrieval models over four standard collections when using predicted term recall as term weights. Third, the proposed method is much more efficient than previous work on query term weight prediction, i.e., term recall weight, which requires an initial retrieval and a local SVD for every new incoming query to obtain features for predicting term recall [22]. The proposed framework derives feature vectors directly from precomputed distributed word vectors; simple computations are sufficient for predicting term weights for new queries.

The remainder of this paper is organized as follows: Section 2 introduces prior research related to query term weighting and term recall weight. Section 3 discusses the preliminaries for term recall prediction and distributed word vectors. Section 4 formally presents our approach in term weight modeling and estimation. Section 5 describes the data sets and the experimental settings. Experimental results as well as data analysis are presented in Section 6. At last, we conclude the paper and discuss some future work in Section 7.

2. RELATED WORK

Query term weighting has been extensively studied in IR literature and a retrieval model reflects its choice of query term weights used. Conceptually, any retrieval model can be abstracted as the following scoring function:

$$Score(q, D) = \sum_{t \in q \cap D} w(t) f(t, D) \qquad (1)$$

where $f(t, D)$ is the *matching score* of term t and document D (for example, term frequency), and $w(t)$, the query term weight, which doesn't specifically depend on D (for example, inverse term frequency), is the quantity we are interested in this paper. By formulating this way, existing retrieval models make different choices about term weight. The most commonly used query term weights in the literature is idf, for example, the vector space model, language model [21], BM25, etc.

Another well known term weight, the term recall weight, is closely related to idf and also attracts broad attention. It is originally captured by the Binary Independence Model (BIM)[16] to emphasize the importance of query term weights, as shown below:

$$Score(q, D) \propto \sum_{t_i \in q \cap D} \log \left(\frac{\mathbb{P}(t_i|R_q)}{1 - \mathbb{P}(t_i|R_q)} \times \frac{1 - \mathbb{P}(t_i|\bar{R}_q)}{\mathbb{P}(t_i|\bar{R}_q)} \right) \qquad (2)$$

where R_q is the set of relevant documents, $\bar{R}_q$ is the set of non-relevant documents, d is a document to be ranked, q is the query, and t_i is a query term. The probability $\mathbb{P}(t|R_q)$ provides one way to weight query terms, known as the RSJ term weight, to improve retrieval performance. However, due to involvement of the relevant set of documents of query q, it is hard to give a reliable estimation of $\mathbb{P}(t|R_q)$ when relevance information is unavailable. Indeed, researchers recognized that the use of term recall weight could lead to huge retrieval gain as $\mathbb{P}(t|R_q)$ is actually the only term about relevance in the ranking function [22], and proposed several ways to predict it.

Croft and Harper [5] modeled the query term weight as a tuned constant (the Croft/Harper Combination Match model). Greiff [6] tried to predict term weight and $\mathbb{P}(t|\bar{R}_q)$ with a linear function of idf. His experiments showed some improvement over the BIM model. More recently, Metzler [12] modeled term weight as a linear function of document frequency. The above modeling of term weight only used df or idf features. The predictions were inadequate as they did not reflect the insight that $\mathbb{P}(t|R_q)$ is a query dependent quantity and that query dependent features are needed to estimate term weight. Recently, Zhao et al. [22] proposed a framework to construct features for query terms from pseudo relevance feedback [20] and use them to predict term weight.

Retrieval models that capture term dependencies have attracted research attention and also demonstrated their retrieval effectiveness compared to unigram query models. One widely used model is the sequential dependency model (SD) [10], which features three types of *query concepts*: terms, bigrams and proximity expression. An example of a sequential dependency query in the Indri query language for the bag-of-words query `apple pie recipe` is:

```
#weight( 0.8 #combine( apple pie recipe )
         0.1 #combine( #1(apple pie)
                       #1(pie recipe) )
         0.1 #combine( #uw8(apple pie)
                       #uw8(pie recipe) )
)
```

#1(apple pie) matches when `apple` and `pie` form a bigram. #uw8(apple pie) matches when `apple` and `pie` occur in any order within a window of 8 words. And #combine is a probabilistic AND operator.

The sequential dependence model provides basic weighting for different types of query concepts. Broad empirical results have validated the effectiveness of SD over the unweighted bag-of-words query model [1, 2, 10]. However, concepts of the same type share the same weight, which is not optimal. Recent research proposed to predict term weights for each concept using features computed from collection statistics, an adaptive model, and an optimization goal of maximizing an evaluation metric such as Mean Average Precision (MAP) [1, 2]. Term weighting strategy emphasizing query aspects is also proposed [23].

Our approach differs from the above methods in that we represent terms and queries using distributed word vectors in a semantic vector space learnt from a global corpus and we construct novel feature vectors from the distributed representations to automatically learn term weights for efficient retrieval.

3. PRELIMINARIES

In this section, we briefly introduce recent work on distributed representations learning from neural network language models and also defines the target term weights we use in our framework.

3.1 Distributed word vectors

Neural network based language models aim to learn the word vector representations and a statistical language model for the underlying text. These models can be mainly attributed to two categories. Models from the first category try to learn the word vector representations and the language model jointly. One example is the neural network language model (NNLM) [3], where a linear projection layer and a non-linear hidden layer are adopted to form a feedforward neural network. Models in the second category learn the word vector representations first and then train the language model with the word vectors. For example, Mikolov, et al. [14] used one neural network with a single hidden layer to learn word vector representations and then trained an N-gram language model on the word vectors. Typically the simple structure of the neural networks used by the second approach makes it less computationally expensive, thus we confine the discussion to this type of system.

Recently, Mikolov et al. [13] proposed two new models, the Continuous Bag-of-Words model (CBOW) and the Continuous Skip-gram model, that have greater training efficiency. CBOW tries to maximize classification of a word by building a log-linear classifier with word vectors of several history words and future words around that location as input, while the Continuous Skip-gram model tries to predict words within a range before and after the current word given the word vector of the current word. Both CBOW and the Continuous Skip-gram model have only one projection layer between the input and output layer without any hidden layer, which significantly reduces the computational cost caused by the non-linear hidden layers in prior neural network based language models. Negative sampling is used to learn the CBOW and the Continuous Skip-gram models [15]. Word vector representations on a Google News corpus with 100 billion words for a vocabulary of 3 million words can be learned in less than one day using modest hardware.

Word vectors learned by both models have performed well in several semantic related task evaluations [13]. Mikolov et al. released the software for training CBOW and Continuous Skip-gram models and a set of pre-trained 300-dimensional word vector representations on the above mentioned Google News corpus[2]. Table 1 presents an example of the 10 closest words to '`Chinese river`' in terms of cosine similarity in the word vector representation space.

Table 1: Example of closest n-gram terms to the phrase '`Chinese river`'.

Word	Cosine similarity
`Yangtze_River`	0.667376
`Yangtze`	0.644091
`Qiantang_River`	0.632979
`Yangtze_tributary`	0.623527
`Xiangjiang_River`	0.615482
`Huangpu_River`	0.604726
`Hanjiang_River`	0.598110
`Yangtze_river`	0.597621
`Hongze_Lake`	0.594108
`Yangtse`	0.593442

It can be seen that the neighbors given by word vectors are indeed semantically related to the input. Also, in this example, unlike `Yangtze_River`, the phrase `Chinese_river` does not belong to the vocabulary of the model; there is no word vector representation for `Chinese_river`. Instead, word vectors for both `Chinese` and `river` are fetched and **averaged** to represent `Chinese_river` in the search for the closest neighbors, which yields meaningful results. A recent analysis [8] justifies the above results by showing that the learning of distributed representations is essentially factorizing a word-context matrix which ensures that words sharing similar context (thus similar meaning) will have similar vector representations.

This ***word vector addition property*** *is the key motivation to represent query and terms as word vectors and thus to derive term features from them to predict target term weights.* See [15] for more examples of word vector addition property. In this paper, we adopt the CBOW framework to learn word vectors and use them to build a term weight prediction framework. The proposed framework can also be applied to word vectors learned by the Continuous Skip-gram model, which we leave as future work.

[2]`https://code.google.com/p/word2vec/`

3.2 Target term weights

As discussed in Section 1, proper query term weights reflects the relative importance of a term to a query and should also help improve retrieval performance. We choose *term recall* weight as our target term weight to predict in our framework for its simplicity to compute and also great potential to improve retrieval performance [22]. Given relevance judgments, it can be estimated as $\mathbb{P}(t|R_q) = \frac{|R_{q,t}|}{|R_q|}$, where R_q is the set of relevant documents to q and $R_{q,t} \subseteq R_q$ is the subset of relevant documents that contain term t.

3.3 Term weights and retrieval models

In this section, we present how true or estimated target term weights can be integrated into different retrieval models.

3.3.1 *Probabilistic language model*

One commonly used retrieval model in IR is the language model, often together with Dirichlet smoothing as shown in Eq. (3),

$$Score(q, D) = \sum_{t \in q \cap D} \log \frac{tf_{t,D} + \mu \frac{cf_t}{|C|}}{|D| + \mu} \tag{3}$$

where tf_t is the term frequency of term t in document D, cf_t is the collection frequency of t, $|D|$ is the length of document D, $|C|$ is the total length of the collection and μ is the smoothing parameter.

True term recall weight or term recall weight estimates can used by a language model in similar fashion as [9], as shwon in Eq. (4).

$$Score(q, D) = \sum_{t \in q \cap D} \left(\mathbb{P}(t|R) \cdot \log \frac{tf_{t,D} + \mu \frac{cf_t}{|C|}}{|D| + \mu} \right) \tag{4}$$

The term recall weighted language model shown above is equivalent to the relevance model under the assumption of binary term occurrences and uniform document length [22].

3.3.2 *BM25*

Another widely adopted retrieval model is BM25, as shown in Eq. (5)

$$Score(q, D) = \sum_{t \in q \cap D} \log \left(\frac{N - df_t + 0.5}{df_t + 0.5} \right) \cdot \frac{tf_{t,D} \cdot (k_1 + 1)}{tf_{t,D} + k_1(1 - b + b\frac{|D|}{avgdl})} \tag{5}$$

where df_t is the document frequency of term t, $avgdl$ is the average document length over the collection and k_1 and b are free parameters. BM25 is actually more directly connected to term recall weight, as the above original BM25 model is an extension of the BIM [16]. By inserting the recall weight estimates to BM25, we get the recall-enhanced BM25 as shown in Eq. (6).

$$Score(q, D) = \sum_{t \in q \cap D} \log \left(\frac{\mathbb{P}(t|R)}{1 - \mathbb{P}(t|R)} \cdot \frac{N - df_t + 0.5}{df_t + 0.5} \right) \cdot \frac{tf_{t,D} \cdot (k_1 + 1)}{tf_{t,D} + k_1(1 - b + b\frac{|D|}{avgdl})}. \tag{6}$$

4. TERM WEIGHTS LEARNING WITH DISTRIBUTED WORD VECTORS

In this section, we present our model for estimating term weights with distributed word vectors.

Table 2: Notations

M	number of queries
N	total number of terms in all M queries
n_i	number of terms in the ith query
q_i	the ith query
t_{ij}	the jth term of the ith query
r_{ij}	true term weight for t_{ij}
p	dimension of distributed word vector
$\boldsymbol{w}_{ij}$	distributed word vector for t_{ij}
$\boldsymbol{x}_{ij}$	feature vector for t_{ij}
$\boldsymbol{X}$	feature matrix
$\boldsymbol{y}$	regression labels vector
λ	regularization parameter
$\boldsymbol{\beta}$	feature weights vector

Suppose we have a set of M queries $Q = \{q_1, q_2, ..., q_M\}$ and each query q_i has n_i terms for $i = 1, 2, ..., M$. Let t_{ij} represent the jth term of query q_i for $j = 1, 2, ..., n_i$, r_{ij} denote the true term weight estimated from relevance judgments for t_{ij}, hence $r_{ij} \in [0, 1]$ and $N = \sum_{i=1}^{M} n_i$ denote the total number of query terms. (Refer to Table 2 for a summary of notations.)

Let $\boldsymbol{w}_{ij} \in \mathbb{R}^p$ denote the continuous distributed word vector representation for term t_{ij}, where p is the dimension of the word vector. In this paper, we propose to directly construct the feature vector $\boldsymbol{x}_{ij}$ for term t_{ij} from the distributed word vector representations of term t_{ij} and other terms in the same query q_i as

$$\boldsymbol{x}_{ij} = \boldsymbol{w}_{ij} - \overline{\boldsymbol{w}}_{q_i} \tag{7}$$

where

$$\overline{\boldsymbol{w}}_{q_i} = \frac{1}{n_i} \sum_{k}^{n_i} \boldsymbol{w}_{ik} \tag{8}$$

is the mean of all word vectors of terms in q_i. Hence, the feature vectors have the same dimension as the word vectors. An naive example of feature vector construction with $p = 2$ is illustrated in Figure 1.

Intuitively, proper term weights measures the relative importance of a term w.r.t. the whole query (hence w.r.t. the other terms in the same query). In other words, a term with a higher term weight means that it's more important for the term to represent the meaning of the query. Meanwhile, the feature vector of a term defined above is the difference of the term to the center of distributed representations for all terms in the query, which also serves as the vector representation for the entire query (by applying the word vector addition properties). Hence, *the feature vector measures the semantic difference of a term to the whole query.* Believing that the above two measures are somewhat related, we propose to construct features above and learn a model to map from the feature vectors to term weight labels.

As a side note, we can see that the above constructed feature vectors are hence query dependent, which is also necessary in that term weight, from its definition, is also query dependent. As we will see in the experiments, the above construction is simple and yet effective.

Having defined the features, we now turn to formulate the model to map from feature to term weight labels. As the

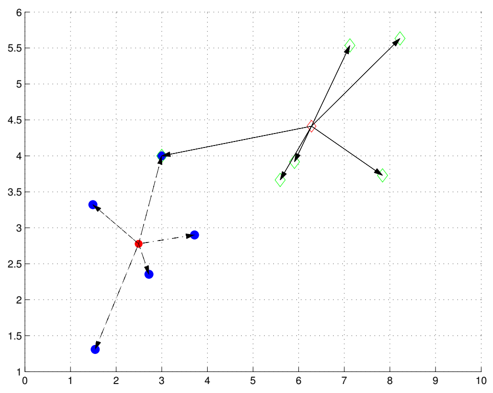

Figure 1: Demonstration of transforming global 2-dimensional word vectors into feature vectors local to the query. In the graph, blue solid circles represent terms in one query and green diamonds represent terms in another query. Note that the two queries share one common term located at (3, 4). The red solid circle represents the center for the circles, and the red diamond represents the center for the diamonds. The dashed and solid arrows are then the transformed feature vectors for all the terms.

adopted target term weight is a probability, we first map it from $(0, 1)$ onto the real line $\mathbb{R}$ with a logit function to avoid numerical issues, as shown below.

$$y_{ij} = \text{logit}(r_{ij}) = \log \frac{r_{ij}}{1 - r_{ij}} \tag{9}$$

We then formulate the "transformed" term weight y as a linear function of the term feature vectors defined above, that is $y = \boldsymbol{\beta}^\top \boldsymbol{x}$, and employ ℓ_1-norm regularization to learn the feature weights $\boldsymbol{\beta}$ (We also tried ℓ_2-norm regularization and ℓ_1-norm regularization works a little better). Formally, the optimization problem is

$$\hat{\boldsymbol{\beta}} = \arg \min_{\boldsymbol{\beta} \in \mathbb{R}^p} \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{n_i} (y_{ij} - \boldsymbol{\beta}^\top \boldsymbol{x}_{ij})^2 + \lambda \|\boldsymbol{\beta}\|_1 \tag{10}$$

where $\lambda \geq 0$ is the regularization parameter controlling the balance between prediction error on training data and model complexity, and needs to be tuned by cross validation. In matrix form, the above optimization problem can be expressed as

$$\hat{\boldsymbol{\beta}} = \arg \min_{\boldsymbol{\beta} \in \mathbb{R}^p} \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 \tag{11}$$

where $\boldsymbol{y} \in \mathbb{R}^N$ is

$$\boldsymbol{y} = (y_{11}, y_{12}, ..., y_{1n_1}, y_{21}, y_{22}, ...y_{2n_2}, ..., y_{M1}, y_{M2}, ..., y_{Mn_M})^\top \tag{12}$$

is target term weight vector with term weights of all terms in the training queries stacked, and $\boldsymbol{X} \in \mathbb{R}^{N \times p}$ is

$$\boldsymbol{X} = (\boldsymbol{x}_{11}, \boldsymbol{x}_{12}, ..., \boldsymbol{x}_{1n_1}, \boldsymbol{x}_{21}, \boldsymbol{x}_{22}, ...\boldsymbol{x}_{2n_2}, ..., \boldsymbol{x}_{M1}, \boldsymbol{x}_{M2}, ..., \boldsymbol{x}_{Mn_M})^\top \tag{13}$$

It is easy to verify that optimization problem (11) is equivalent to (10) and is of standard form of LASSO regression [19].

Specifically, the subgradient of the objective function w.r.t $\boldsymbol{\beta}$ in problem (11) is

$$\frac{\partial f(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}) + \lambda \boldsymbol{s} \tag{14}$$

where

$$s_i \in \begin{cases} \operatorname{sign}(\beta_i) & \text{if } \beta_i \neq 0 \\ [-1, 1] & \text{if } \beta_i = 0 \end{cases} \tag{15}$$

Efficient gradient based methods can be applied to solve the above problem and after we get the optimum $\hat{\boldsymbol{\beta}}$, when a new query term t_* with feature vector $\boldsymbol{x}_*$ arrives, we predict its term weight as

$$\widehat{\mathbb{P}(t_*|R)} = \operatorname{sigmoid}\left(\hat{\boldsymbol{\beta}}^\top \boldsymbol{x}_*\right) = \frac{\exp\left(\hat{\boldsymbol{\beta}}^\top \boldsymbol{x}_*\right)}{1 + \exp\left(\hat{\boldsymbol{\beta}}^\top \boldsymbol{x}_*\right)} \tag{16}$$

We refer to the above proposed method as `DeepTR` for it using distributed representations of words to model term weights.

`DeepTR` is a very efficient method of predicting term weights, which enables it to be used in online services where latency (the time required to respond to a query) must be kept low. The word vectors and the regression model are trained offline. Predicting $\mathbb{P}(t_{ij}|R)$ for a new query involves loading the word vectors for the query terms, transforming them into feature vectors (averaging and subtraction), an inner product with the learned feature weights, and a sigmoid function. This is far more efficient than some prior methods that were effective but computationally complex (e.g., [22]).

After we gain estimates of the term weights, we construct the following query model variations in Indri query language (with the above `apple pie recipe` keyword query as an example):

- `DeepTR-BOW`:

```
#weight( $\widehat{\mathbb{P}}$(apple|$R$) apple
         $\widehat{\mathbb{P}}$(pie|$R$) pie
         $\widehat{\mathbb{P}}$(recipe|$R$) recipe )
)
```

- `DeepTR-SD`:

```
#weight(
0.8 #weight( $\widehat{\mathbb{P}}$(apple|$R$) apple
             $\widehat{\mathbb{P}}$(pie|$R$) pie
             $\widehat{\mathbb{P}}$(recipe|$R$) recipe
0.1 #combine(#1(apple pie)
             #1(pie recipe) )
0.1 #combine(
             #uw8(apple pie)
             #uw8(pie recipe) )
)
```

Note that in `DeepTR-SD`, we mainly focus on reweighting the unigrams, as unigram reweighting plays a much more significant role than bigrams and proximity expressions in improving retrieval accuracies, which is also confirmed by previous studies [22, 1, 2]. The proposed framework can be directly extended to bigrams, proximity expressions and other query concepts that can be represented by a standard inverted list containing positions, however the training data are more sparse for more complex concepts, thus we leave that for future study.

The two query model variations proposed above can be used with language modeling and BM25 retrieval models; in the next section, we present experiments with both types of retrieval model.

5. EXPERIMENTAL METHODOLOGY

This section describes how we evaluated our work experimentally. We conduct experiments on 4 TREC test collections consisting of one Robust track dataset and three Web Track datasets. The document collections differ in size, from a small and traditional TREC Robust Track collection to large Web Track collections of web documents. The dataset sizes and the queries used with each dataset are shown in Table 3. The first three datasets are standard datasets used without change. The ClueWeb09B dataset is the standard ClueWeb09 Category B dataset after spam documents are removed. The Waterloo Fusion spam scores[3] were used, and the filtering threshold was set to 70%.

Table 3: Collections and topics. Spam documents were removed from the ClueWeb09 Category B dataset (Waterloo Fusion spam scores, 70% threshold).

Collection	# Docs	# Words	TREC Topics
ROBUST04	528K	253M	351-450, 601-700
WT10g	1,692K	1,076M	451-550
GOV2	25,205K	24,007M	701-850
ClueWeb09B	29,038K	23,890M	1-200

We use the Indri search engine[4] to index the corpus. The Krovetz stemmer was used on queries and documents. A standard list of English stop words is maintained for query processing. Additionally, for thee web track datasets, anchor texts from in-links is treated as part of the document and hence is indexed. We use descriptions of the TREC topics as queries stopped by a list of standard stop words and several *stop phrases* such as “find information”.

We use `DeepTR-BOW` to denote the re-weighted keyword queries and `DeepTR-SD` to denote the re-weighted sequential dependency queries. All of the above query models can be expressed in the Indri query language. When constructing the sequential dependency model queries, we use weights 0.8, 0.1 and 0.1 for terms, bigrams and unordered windows, respectively as they have shown to be effective in prior research and practical applications [10, 11].

For retrieval, we use a language model with Dirichlet smoothing [21] and BM25 to test both types of weighted queries. Although sequential dependency model queries are not typically used with the BM25 retrieval model, they are not incompatible with BM25. It is straightforward to calculate the *tf* and *df* statistics that BM25 needs in order to calculate scores for bigrams (`#1(apple pie)`) and proximity expressions (`#uw8(apple pie)`) [2]. We show the results for these query concepts with BM25 to demonstrate the effectiveness of our method. A minor change is made to Indri to allow for the insertion of term weight estimates to the BM25 retrieval model, as shown in Eqn. (6). Retrieval parameters are all set to Indri default values, which is $\mu = 2500$ for language model and $k_1 = 1.2, b = 0.75$ for BM25.

[3]`http://plg.uwaterloo.ca/~gvcormac/clueweb09spam`
[4]`http://lemurproject.org/indri/`

For `DeepTR`, we use several sets of distributed word vector representations. The first one is the pre-trained 300-dimensional vectors released by Google, which is trained on a Google News corpus of about 100 billion words.[5] The second set of word vectors was trained on a subset of the ClueWeb09 Category B corpus.[6] Spam documents were removed, as described above; the remaining documents were processed by Boilerpipe[7] to remove the "unimportant" parts of each page. Mikolov's Continuous Bag-of-Words Model software [13, 15] was used to learn the word vectors with 100, 300, 500 dimensions, repectively. We also trained word vectors on the ROBUST04 and WT10g collections with 300 dimensions to enable comparison of corpus-specific word vectors trained from small datasets with corpus-independent word vectors trained on larger web datasets.

We train `DeepTR` on the set of queries with 5-fold cross validation and report the averaged performance. That is, we divide the queries to 5 folds and on each fold, we train the model using three of them, validates the performance on one fold of them as development set to pick the regularization parameter λ from $\{0, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100\}$ and report the performance on the rest fold as testing data. After the model is trained on each fold, term weights for queries in the testing fold are predicted and retrieval performances for the reweighted queries are reported.

We use the `trec_eval`[8] tool provided by TREC to assess the retrieval results. We focus mainly on the Mean Average Precision (MAP) and Precision at 10 (P@10) metrics. For web collections such as GOV2 and ClueWeb09B, where graded relavence judgments are available, we also report Normalized Discounted Cumulative Gain (NDCG) at 20 and Expected Reciprocal Rank (ERR)[4] at 20 values using the `gdeval.pl`[9] tool provided by TREC. Statistical significance of model differences in terms of retrieval performance is judged by a two-sided paired randomization test with $\alpha < 0.05$ rather than the Wilcoxon signed rank test that was used in prior research [1], which is prone to type I errors and considered not reliable for ad hoc IR [17, 18].

We compare the retrieval performance of `DeepTR` to three baseline query models. The first is the unstructured bag-of-words query model (`BOW`). The second is the original sequential dependency model (`SD`) [10]. The third is the weighted sequential dependence model (`WSD`) [1], which is the state of the art model for query reweighting. For `WSD`, we implemented the model with coordinate ascent [11] to directly optimize MAP on the training queries for `WSD`. However, we got slightly worse results than reported in [1, 2], possibly due to that we didn't have the MSN query log feature used in the original paper. As we use an overlapping set of collections and set of queries, and our `BOW` and `SD` baseline results are very close to the ones reported in the `WSD` paper, we decide to show and compare with the performance numbers of `WSD` reported in the original paper [1, 2]. We also implemented the method proposed by Zhao et al. [22] to predict term recall, but the experimental results on our datasets were worse than our unweighted bag-of-words query baselines, thus we omit the comparison from this paper.

[5]`https://code.google.com/p/word2vec/`

[6]`http://lemurproject.org/clueweb09/`

[7]`https://code.google.com/p/boilerpipe/`

[8]`http://trec.nist.gov/trec_eval/trec_eval_latest.tar.gz`

[9]`http://trec.nist.gov/data/web/10/gdeval.pl`

6. EXPERIMENTAL RESULTS

This section presents the results of experiments that compare the `DeepTR` method of setting term weights in two retrieval models (language models, BM25) to three baseline methods (unweighted queries, sequential dependency models, weighted sequential dependency models); the effects of word vectors of varying dimensionality; and the effects of word vectors trained from different types of data.

6.1 Retrieval results with Language Model

The first experiment compared query term weights provided by `DeepTR` in the language model retrieval model to three baseline methods (unweighted queries, sequential dependency models, weighted sequential dependency models). Experimental results are shown in Table 4 for both `DeepTR-BOW` and `DeepTR-SD`. In this experiment, we fix the dimension of word vectors to be 300 and compare the performances of the proposed models with different sources of word vectors.

`DeepTR-BOW` term weights perform better than the unweighted bag-of-words query model (`BOW`) over all collections, significantly outperforming `BOW` on ROBUST04 and GOV2 in terms of MAP and on ROBUST 04 in terms of P@10. `DeepTR-BOW` even achieves higher MAP than the sequential dependency model (`SD`) on WT10g, however the results are not statistically significant. `DeepTR-BOW` gains comparable MAPs to `WSD` in ROBUST04, WT10g and ClueWeb09B. This suggests that `DeepTR-BOW` which models no bigrams and proximity expressions is able to achieve comparable retrieval performances with the sequential dependency models (both unweighted and weighted).

The `DeepTR-SD` query model performs better than the standard sequential dependency model over all collections with all sources of word vectors. Particularly, significant differences are observed on ROBUST04, WT10g and GOV2 with all sets of word vectors; `DeepTR-SD` with word vectors trained on the Google News corpus achieves significantly higher MAPs over `BOW` and `SD` on all four collections. The significant gains of `DeepTR-SD` with respect to `SD` range from 4.2% to 12.1% in MAP. Similar significant improvements for P@10 are also observed. While we cannot draw statistical significance conclusions as we have no performance on individual query of `WSD`, on ROBUST04 and WT10g, `DeepTR-SD` attains better MAPs than `WSD`. This validates the effectiveness of using distributed representations of words as feature sources to predict term weights in improving both overall retrieval performance and top-rank results over the baseline models.

6.2 Retrieval results with BM25

We also show the retrieval results with BM25 in Table 5 for both`DeepTR-BOW` and `DeepTR-SD` to demonstrate the boost in retrieval performance with different retrieval models. Prior research on weighted sequential dependency models [1] did not publish results for BM25, so we are unable to compare against that baseline in this experiment.

`DeepTR-BOW` term weights perform better than the unweighted bag-of-words query model (`BOW`) over all collections, significantly outperforming `BOW` for MAP over all data sets and all word vector variations except for ClueWeb09B with the Google word vectors; although not statistically significant, `DeepTR-BOW` also performs better than `SD` with all word vectors settings on ROBUST04, WT10g and GOV2.

Table 4: Retrieval performance of DeepTR-BOW using language model retrieval. In the first column, the parenthesis indicates the word vector source. Other parentheses show improvements over BOW and SD, respectively. (For WSD, there were no P@10 values reported and no results on ClueWeb09B were reported in [2])

Query Model	ROBUST04		WT10g		GOV2		ClueWeb09B	
	P@10	MAP	P@10	MAP	P@10	MAP	P@10	MAP
`BOW`	0.4245	0.2512	0.3290	0.1943	0.5054	0.2488	0.2667	0.0702
`SD`	0.4414	0.2643	0.3400	0.2032	0.5342	0.2688	0.2798	0.0745
`WSD` (Table 7 in [2])	-	0.2749	-	0.2260	-	0.2946	-	-
`DeepTR-BOW` `(Corpus-specific 300)`	0.4430^b	0.2591^b *(+3.2/-1.9)*	0.3280	0.2103 *(+8.2/+3.5)*	0.5208	0.2646^b *(+6.3/-1.6)*	0.2682	0.0718 *(+2.2/-3.6)*
`DeepTR-BOW` `(GOV2 300)`	0.4430^b	0.2650^b *(+5.5/+0.3)*	0.3330	0.2111^b *(+8.7/+3.9)*	0.5208	0.2646^b *(+6.3/-1.6)*	0.2667	0.0741 *(+5.6/-0.5)*
`DeepTR-BOW` `(ClueWeb09B 300)`	0.4454^b	0.2657^b *(+5.8/+0.5)*	0.3270	0.2129 *(+9.6/+4.8)*	0.5121	0.2685^b *(+7.9/-0.1)*	0.2682	0.0718 *(+2.2/-3.6)*
`DeepTR-BOW` `(Google 300)`	0.4450^b	0.2673^b *(+6.4/+1.2)*	0.3380	0.2122^b *(+9.3/+4.5)*	0.5221	0.2630^b *(+5.7/-2.2)*	0.2667	0.0732 *(+4.2/-1.8)*
`DeepTR-SD` `(Corpus-specific 300)`	0.4558^b_s	0.2754^b_s *(+9.6/+4.2)*	0.3510	0.2182^b_s *(+12.3/+7.4)*	0.5490^b	0.2831^b_s *(+13.8/+5.3)*	0.2879^b	0.0748 *(+6.5/+0.5)*
`DeepTR-SD` `(GOV2 300)`	0.4610^b_s	0.2781^b_s *(+10.7/+5.2)*	0.3700^b_s	0.2223^b_s *(+14.4/+9.4)*	0.5490^b	0.2831^b_s *(+13.8/+5.3)*	0.2854	0.0806^b_s *(+14.8/+8.2)*
`DeepTR-SD` `(ClueWeb09B 300)`	0.4659^b_s	0.2810^b_s *(+11.9/+6.3)*	0.3610^b_s	0.2279^b_s *(+17.3/+12.1)*	0.5597^b_s	0.2890^b_s *(+16.2/+7.5)*	0.2879^b	0.0748 *(+6.5/+0.5)*
`DeepTR-SD` `(Google 300)`	0.4627^b_s	0.2842^b_s *(+13.1/+7.5)*	0.3560	0.2256^b_s *(+16.1/+11.0)*	0.5497^b	0.2814^b_s *(+13.1/+4.7)*	0.2869	0.0780^b_s *(+11.0/+4.7)*

b : Statistically significant difference with `BOW`
s : Statistically significant difference with `SD`

The `DeepTR-SD` query model performs better than unweighted queries and the standard sequential dependency model, delivering significantly higher MAPs over all collections and all sources of word vectors except for ClueWeb09B with the Google word vectors. The significant gains of `DeepTR-SD` over `SD` range from 1.9% to 7.3%. Similar trends over P@10 are also observed. `DeepTR-SD` with word vectors trained on ClueWeb09B achieves significantly higher P@10 than `SD` on ROBUST04, WT10g and GOV2.

6.3 Word vector dimensionality

Word vectors of different dimensionality provide different levels of granularity that may or may not be useful for setting term weights; they may also require different amounts of training data. Our third experiment investigates the effects of word vectors containing 100, 300, and 500 dimensions on term weights produced for language models. The ClueWeb09B corpus is used for these experiments due to its size, availability, and strong performance in the experiments above. Experimental results for `DeepTR-BOW` and `DeepTR-SD` are shown d together in Table 6.

Word vectors of 100 dimensions work best for unstructured BOW queries on ROBUST04, WT10g and ClueWeb09B, while the best MAP for GOV2 is achieved with word vectors of 300 dimensions. Similar trends are observed for the `DeepTR-SD` query model. Notably, `DeepTR-SD` with word vectors of 100 dimensions attains a significant 7.9% MAP improvement over `SD` on ROBUST04, and a significant 16.8% MAP improvement over SD on WT10g. On GOV2, word vectors of 300 dimensions help `DeepTR-SD` significantly improve over `SD` by 7.5%; last on ClueWeb09B, word vectors of 500 dimensions achieve the best result over all three dimensions, by a 4.9% over `SD`. We also observe similar results on P@10 when varying the dimensionality of word vectors. We also evaluated the effects of word vector dimensionality on term weights for BM25 retrieval and trends are similar to those for language model retrieval; the results are omitted due to space constraints.

Our results suggest that 100 dimensions is sufficient for estimating very effective term weights.

6.4 Corpus effects

We turn back to Tables 4 and 5 to compare retrieval results obtained with corpus-specific word vectors and word vectors trained from larger, more general, and external corpora (GOV2, ClueWeb09B, and Google News).

`DeepTR-BOW` performed about equally well with all three external corpora; the differences among them were too small and inconsistent to support any conclusion about which is best. However, although no external corpus was best for all datasets, the language model and BM25 retrieval models agreed on which source of word vectors was best for a particular corpus. This agreement suggests that the learned term weights are independent of a particular retrieval model, which was the purpose of training with term weight values.

The corpus-specific word vectors were never best in these experiments, even for GOV2 and ClueWeb09, which are large and provided 'best' performance for other datasets. However, given the wide range of training data sizes – varying from 250 million words to 100 billion words – it is striking how little correlation there is between search accuracy and the amount of training data.

Similar trends are observed for the `DeepTR-SD` query models.

6.5 Robustness analysis

Figure 2 presents the detailed number of queries improved or hurt by proposed method using word vectors trained from various sources compared to the LM baseline. It's clear that for all methods using the predicted term weights, the numbers of improved queries are significantly larger than that of hurt queries. Moreover, when compared to sequential dependency model (`SD`), all the methods using the estimated weights help more and hurt fewer queries than sequential dependency model (`SD`), especially in the [-20%, 0) range as shown in the figure.

Table 5: Retrieval performance of DeepTR-BOW and DeepTR-SD using BM25 retrieval. In the first column, the parenthesis indicates the word vector source. (No WSD results are reported as no BM25 experiments are conducted in [2])

Query Model	ROBUST04		WT10g		GOV2		ClueWeb09B	
	P@10	MAP	P@10	MAP	P@10	MAP	P@10	MAP
BOW	0.4189	0.2460	0.3330	0.1783	0.4926	0.2334	0.2030	0.0566
SD	0.4394	0.2546	0.3350	0.1817	0.5215	0.2409	0.2030	0.0600
DeepTR-BOW (Corpus-specific 300)	0.4341^b	0.2566^b *(+4.3/+0.8)*	0.3280	0.1903^b *(+6.7/+4.7)*	0.5107	0.2430^b *(+4.1/+0.9)*	0.2086	0.0593^b *(+4.7/-1.2)*
DeepTR-BOW (GOV2 300)	0.4390^b	0.2561^b *(+4.1/+0.6)*	0.3340	0.1910^b *(+7.1/+5.1)*	0.5107	0.2430^b *(+4.1/+0.9)*	0.2136^b	0.0596^b *(+5.3/-0.7)*
DeepTR-BOW (ClueWeb09B 300)	0.4426^b	0.2590^b *(+5.3/+1.8)*	0.3390	0.1932^b *(+8.3/+6.3)*	0.5081	0.2462^b *(+5.5/+2.2)*	0.2086	0.0593^b *(+4.7/-1.2)*
DeepTR-BOW (Google 300)	0.4382^b	0.2600^b *(+5.7/+2.1)*	0.3450	0.1922^b *(+7.8/+5.7)*	0.5181^b	0.2449^b *(+4.9/+1.7)*	0.2091	0.0584 *(+3.1/-2.7)*
DeepTR-SD (Corpus-specific 300)	0.4406^b	0.2595^b_s *(+5.5/+1.9)*	0.3470_s	0.1944^b_s *(+9.0/+7.0)*	0.5356^b_s	0.2478^b_s *(+6.1/+2.9)*	0.2066	0.0613^b_s *(+8.2/+2.1)*
DeepTR-SD (GOV2 300)	0.4450^b	0.2609^b_s *(+6.1/+2.5)*	0.3500_s	0.1941_s *(+8.8/+6.8)*	0.5356^b_s	0.2478^b_s *(+6.1/+2.9)*	0.2091	0.0616^b_s *(+8.8/+2.6)*
DeepTR-SD (ClueWeb09B 300)	0.4486^b_s	0.2630^b_s *(+6.9/+3.3)*	0.3510_s	0.1951^b_s *(+9.4/+7.3)*	0.5349^b_s	0.2502^b_s *(+7.2/+3.9)*	0.2066	0.0613^b_s *(+8.2/+2.1)*
DeepTR-SD (Google 300)	0.4458^b	0.2627^b_s *(+6.8/+3.2)*	0.3510_s	0.1938_s *(+8.7/+6.7)*	0.5322^b	0.2461^b *(+5.4/+2.2)*	0.2030	0.0604 *(+6.8/+0.7)*

b : Statistically significant difference with BOW
s : Statistically significant difference with SD

Table 6: Retrieval performance of DeepTR-BOW and DeepTR-SD using language model retrieval. In the first column, the parenthesis indicates the word vector source. (For WSD, there were no P@10 values reported and no results on ClueWeb09B were reported in [2])

Query Model	ROBUST04		WT10g		GOV2		ClueWeb09B	
	P@10	MAP	P@10	MAP	P@10	MAP	P@10	MAP
BOW	0.4245	0.2512	0.3290	0.1943	0.5054	0.2488	0.2667	0.0702
SD	0.4414	0.2643	0.3400	0.2032	0.5342	0.2688	0.2798	0.0745
WSD (Table 7 in [2])	-	0.2749	-	0.2260	-	0.2946	-	-
DeepTR-BOW (ClueWeb09B 100)	0.4410^b	0.2681^b *(+6.7/+1.4)*	0.3440	0.2239^b *(+15.3/+10.2)*	0.5228	0.2667^b *(+7.2/-0.8)*	0.2727	0.0727 *(+3.6/-2.4)*
DeepTR-BOW (ClueWeb09B 300)	0.4454^b	0.2657^b *(+5.8/+0.5)*	0.3270	0.2129 *(+9.6/+4.8)*	0.5121	0.2685^b *(+7.9/-0.1)*	0.2682	0.0718 *(+2.2/-3.6)*
DeepTR-BOW (ClueWeb09B 500)	0.4398^b	0.2679^b *(+6.6/+1.4)*	0.3480	0.2179^b *(+12.1/+7.2)*	0.5054	0.2655^b *(+6.7/-1.2)*	0.2707	0.0726 *(+3.4/-2.5)*
DeepTR-SD (ClueWeb09B 100)	0.4711^b_s	0.2851^b_s *(+13.5/+7.9)*	0.3700^b_s	0.2373^b_s *(+22.1/+16.8)*	0.5557^b_s	0.2848^b_s *(+14.4/+5.9)*	0.2874^b	0.0778^b_s *(+10.8/+4.5)*
DeepTR-SD (ClueWeb09B 300)	0.4659^b_s	0.2810^b_s *(+11.9/+6.3)*	0.3610^b_s	0.2279^b_s *(+17.3/+12.1)*	0.5597^b_s	0.2890^b_s *(+16.2/+7.5)*	0.2879^b	0.0748 *(+6.5/+0.5)*
DeepTR-SD (ClueWeb09B 500)	0.4606^b_s	0.2817^b_s *(+12.2/+6.6)*	0.3660^b_s	0.2306^b_s *(+18.7/+13.5)*	0.5550^b	0.2860^b_s *(+14.9/+6.4)*	0.2894^b	0.0781^b_s *(+11.3/+4.9)*

b : Statistically significant difference with BOW
s : Statistically significant difference with SD

6.6 Query length

We investigate how DeepTR-SD performs on queries with different lengths. We divide the queries into three groups: with no more than 3 terms (Len -3), 4 to 5 terms (Len 4-5) and 6 terms or more (Len 6+). Figure 3 shows the relative gains of SD and DeepTR-SD (using vectors pre-trained from Google News) over BOW. It's clear that DeepTR-SD works significantly better than SD on mid-range and long queries (Len 4-5 and Len 6+) on all collections. It's worth noting that SD performs worse than BOW for long queries on WT10g, while in contrast DeepTR-SD improves BOW by 14%. For short queries (Len -3), DeepTR-SD outperforms SD over 3 out of 4 collections, which also renders DeepTR-SD a better alternative for SD.

6.7 Graded relevance

Our final experiment investigates retrieval quality using the graded relevance judgments that are available for web collections such as GOV2 and ClueWeb09B. Table 7 presents the experimental results using the NDCG@20 and ERR@20 metrics that have been widely used in web search scenarios. Similar to prior experiments, we see that DeepTR-SD produces more accurate rankings than BOW and SD on both GOV2 and ClueWeb09.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we present a novel framework for learning term weights using distributed representations of words from the deep learning literature. We motivate the framework by adopting the word vectors to represent terms and further to represent the query due to the ability to represent things semantically of word vectors. Therefore we propose to construct features for terms directly from the word vectors and model the target term weight as a regression problem.

We conducted experiments with two retrieval models, the language model and BM25, over four text collections of vary-

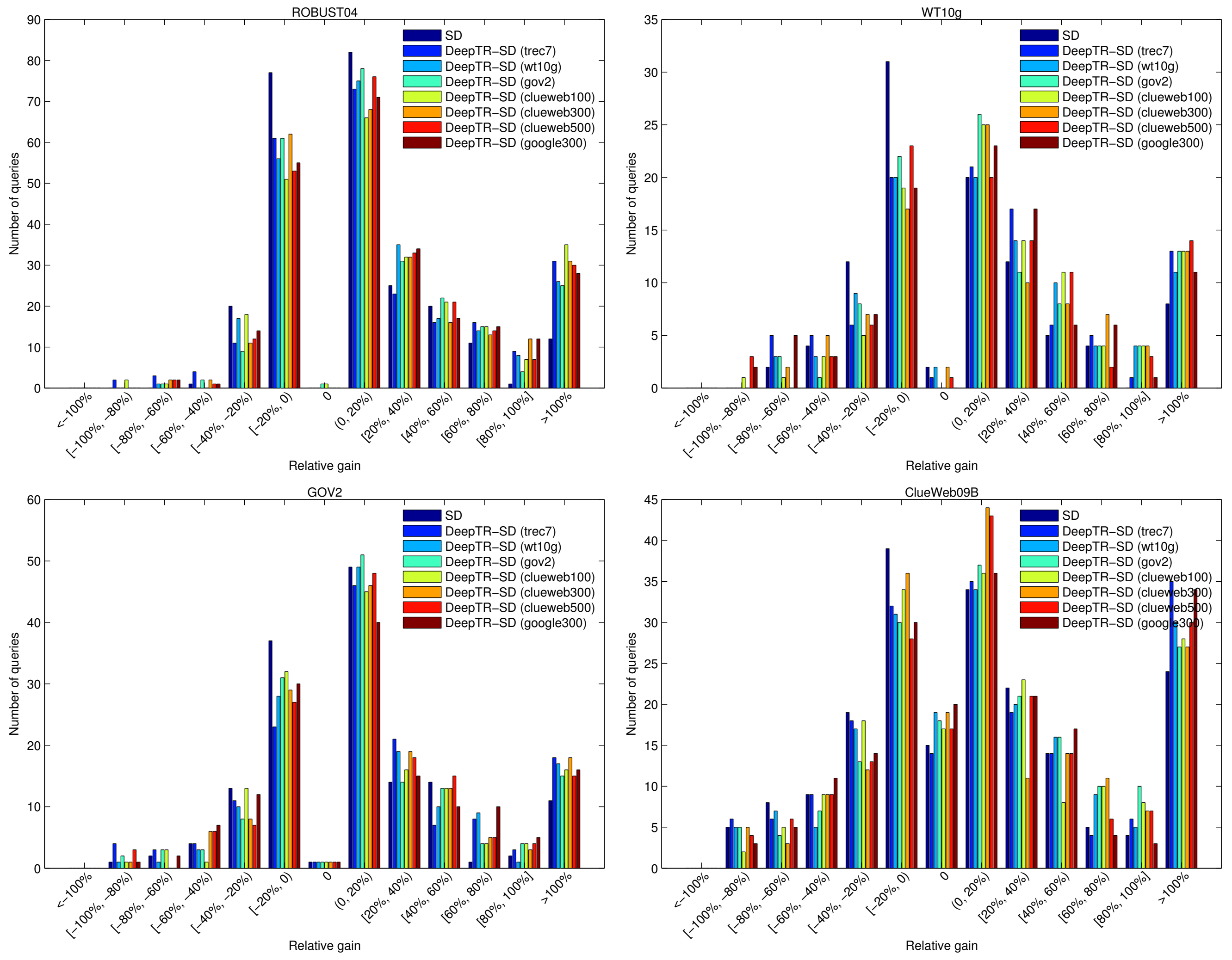

Figure 2: Robustness analysis over all data sets in terms of MAP over LM (best viewed in color)

ing sizes to demonstrate the effectiveness of the proposed framework. Specifically, the proposed framework predicts term weight for query terms that can be used to weight unstructured bag-of-words queries and queries that are a slight variant of the sequential dependency model. We observed significant improvements at high precision and throughout the rankings over the unweighted bag-of-words queries and the original unweighted sequential dependency model queries.

The proposed method is time efficient. Once the word vectors are trained in an offline step (that is itself relatively efficient), online prediction of term weight for new query terms involves only a simple inner product with the learned feature weights.

There are several promising directions for further research. One can extend the method from creating vectors for bigrams and proximity terms by averaging the word vectors of their constituents to a direct modeling of bigrams and proximity terms; although one might expect sparse training data for these query terms to be a problem, our results with small corpora suggest that it may not be necessary to have massive amounts of information for each term. Word vectors may also be useful for identifying terms that should be the focus of query expansion or terms that would be good expansion terms.

Acknowledgements

This work is supported by National Science Foundation under grant IIS-1018317. We thank the anonymous reviewers for their helpful comments.

8. REFERENCES

[1] M. Bendersky, D. Metzler, and W. B. Croft. Learning concept importance using a weighted dependence model. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 31–40. ACM, 2010.

[2] M. Bendersky, D. Metzler, and W. B. Croft. Parameterized concept weighting in verbose queries. In *SIGIR*, pages 605–614, 2011.

[3] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[4] O. Chapelle, D. Metlzer, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. In *CIKM*, pages 621–630, 2009.

Table 7: Retrieval performance of `DeepTR-SD` using graded relevance judgments. Parenthesis indicates the word vector source.

Query Model	Language Model Retrieval				BM25 Retrieval			
	GOV2		ClueWeb09B		GOV2		ClueWeb09B	
	NDCG@20	ERR@20	NDCG@20	ERR@20	NDCG@20	ERR@20	NDCG@20	ERR@20
`BOW`	0.3732	0.1493	0.1597	0.1253	0.3789	0.1487	0.1188	0.1075
`SD`	0.4059	0.1607	0.1694	0.1243	0.3940	0.1554	0.1294	0.1059
`DeepTR-SD (GOV2 300)`	0.4121^b	0.1626^b	0.1743^b	0.1312_s	0.4008^b_s	0.1590^b_s	0.1307	0.1068
`DeepTR-SD (ClueWeb09B 300)`	0.4146^b	0.1614^b	0.1663	0.1255	0.4033^b_s	0.1587^b_s	0.1305	0.1067
`DeepTR-SD (Google 300)`	0.4112^b	0.1600	0.1698	0.1302	0.4010^b_s	0.1586_s	0.1298	0.1050

b : Statistically significant difference with `BOW`
s : Statistically significant difference with `SD`

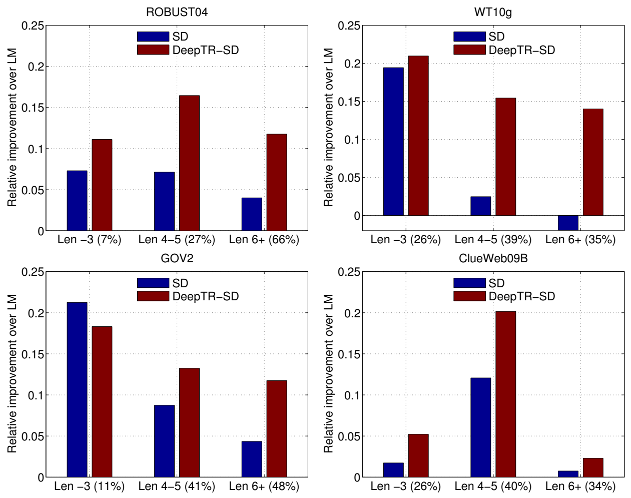

Figure 3: Performance gains of `SD` and `DeepTR-SD` over `BOW` w.r.t to query length. Y-axis shows the relative gain in MAP over `BOW` and parentheses in X-axis present the percentage of queries in each group. All retrievals are performed using LM. (best viewed in color)

[5] W. B. Croft and D. J. Harper. Using probabilistic models of document retrieval without relevance information. *Journal of Documentation*, 35:285–295, 1979.

[6] W. R. Greiff. A theory of term weighting based on exploratory data analysis. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 11–19. ACM Press, 1998.

[7] B. J. Jansen, D. L. Booth, and A. Spink. Determining the user intent of web search engine queries. In *Proceedings of the 16th International Conference on World Wide Web, Banff, Alberta, Canada, May 8-12, 2007*, pages 1149–1150, 2007.

[8] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2177–2185, 2014.

[9] D. Metzler and W. B. Croft. Combining the language model and inference network approaches to retrieval. *Inf. Process. Manage.*, 40(5):735–750, 2004.

[10] D. Metzler and W. B. Croft. A markov random field model for term dependencies. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 472–479. ACM, 2005.

[11] D. Metzler and W. B. Croft. Linear feature-based models for information retrieval. *Information Retrieval*, 10(3):257–274, 2007.

[12] D. Metzler, V. Lavrenko, and W. B. Croft. Formal multiple-bernoulli models for language modeling. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 540–541. ACM, 2004.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[14] T. Mikolov, J. Kopecký, L. Burget, O. Glembek, and J. Cernocký. Neural network based language models for highly inflective languages. In *ICASSP*, pages 4725–4728, 2009.

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[16] S. E. Robertson and K. S. Jones. Relevance weighting of search terms. *Journal of the American Society for Information science*, 27(3):129–146, 1976.

[17] M. Sanderson and J. Zobel. Information retrieval system evaluation: Effort, sensitivity, and reliability. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 162–169, New York, NY, USA, 2005. ACM.

[18] M. D. Smucker, J. Allan, and B. Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 623–632, New York, NY, USA, 2007. ACM.

[19] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

[20] J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *In Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 4–11, 1996.

[21] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of SIGIR*, pages 334–342. ACM, 2001.

[22] L. Zhao and J. Callan. Term necessity prediction. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, pages 259–268. ACM, 2010.

[23] W. Zheng and H. Fang. Query aspect based term weighting regularization in information retrieval. In *Advances in Information Retrieval, 32nd European Conference on IR Research, ECIR 2010, Milton Keynes, UK, March 28-31, 2010. Proceedings*, pages 344–356, 2010.