Word Embedding Based Query Expansion

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Computer Science

by

Amritap Chowdhury
[Roll No: CS1601]

under the guidance of

Dr. Mandar Mitra
Associate Professor
Computer Vision and Pattern Recognition Unit

Indian Statistical Institute
Kolkata-700108, India

July 2018

CERTIFICATE

I certify that I have read the thesis prepared under my guidance by Amritap Chowdhury, entitled **"Word Embedding Based Query Expansion"** and in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Master of Technology in Computer Science of Indian Statistical Institute

Dr. Mandar Mitra
Associate Professor,
Computer Vision and Pattern Recognition Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Acknowledgments

I am sincerely grateful to my supervisor Dr. Mandar Mitra for constantly supporting and encouraging me throughout the dissertation with his patience and knowledge whilst giving me the freedom to work in my own way. He has literally taught me how to do good research and has always motivated and refined my ideas. He has not only been my supervisor but has been an ideal inspiration from whom there are a lot of things to learn about life. He is such a nice person, that he just can't say 'No' to anyone, he is always willing to help. One just cannot wish to have a friendlier and better supervisor than him.

I am deeply thankful to Dr. Pabitra Mitra of IIT Kharagpur for motivating me towards Information Retrieval and Machine Learning and for suggesting me to work under the guidance of Dr. Mandar Mitra.

I would also like to convey my gratitude to all the members of the Information Retrieval lab for making it such a convival place to work in. Specially I would like to thank Dwaipayan Roy for helping me out with Lucene and other technicalities that were required for my dissertation. This year we were not lucky enough to have a formal course in Information Retrieval, but he has personally taken the pain to clear all my doubts and has explained me every single concept whenever I needed, he has really been my IR Guru. I am also thankful to the other members of the lab including Riya Roy, Ayan Bandapadhyay and Suchana Datta for inspiring me in research and life through our interactions during the long hours in the lab.

Last but not the least, I am really grateful to my parents for their everlasting support. They have always had faith in me and have inspired me in every difficult situations of my life.

Amritap Chowdhury
M.Tech Computer Science
Indian Statistical Institute
Kolkata - 700108 , India.

Abstract

Continuous space word embeddings have received a great deal of attention in Information Retrieval for their ability to model term similarity and other relationships. We have studied the use of term relatedness in the context of query expansion for *ad-hoc* information retrieval. In our first approach we have proposed a time efficient retrieval algorithm for query expansion using *pseudo-locally* constrained word embeddings. In our second approach we have tried to present a learning approach that adaptively predicts the balance co-efficients between the original query model and the local and global expansion language models. In this approach we have also tried to predict the optimal number of expansion terms required for the local and global embedding based query expansion methods. In our third approach we have fused the results of query expansion based on local and global embeddings to have an improved performance over both the methods. In all the above approaches, we have performed our experiments on standard TREC *ad-hoc* data(Disk 4 and 5) with query sets TREC 6, 7, 8 and *robust*. Our first and third approaches have shown comparable performance with the state-of-the-art query expansion methods, based on word embeddings, but, our second approach has failed to perform in accordance to our hyposthesis.

Keywords: *Information Retrieval, Query Expansion, Word Embedding.*

Contents

Chapter 1

Brief Overview on Information Retrieval

The history of archieving and finding information from them can be traced back to 3000 BC, when the Sumerians used to store clay tablets with cuneiform inscriptions [1]. Organization and access to archieves has always been critical for efficient use of information. Even back then the Sumerians have developed special classifications to identify every tablet and its content.

With the advancement in the ways of storing information and enhancement in the computational power, accessing and retrieving information from large collections efficiently has become a necessity. In 1945 Vannevar Bush published a ground breaking article titled "As We May Think" that gave birth to the idea of automatic access to large amounts of stored knowledge. In the 1950s, this idea materialized into more concrete descriptions of how archives of text could be searched automatically. Several works emerged in the mid 1950s that elaborated upon the basic idea of searching text with a computer [1]. A lot of Information Retrieval based applications have been developed and commercialized in the last sixty years. Many of them like google web search engine and conversational information seeking agents like alexa and siri has become an integral part of our day to day life.

1.1 What is Information Retrieval

Information Retrieval is a very broad term. From searching for a book in a library management system to searching for a document in the world wide web, seeking of any kind of archieval information is Information Retrieval. It ranges from looking for a phone number in a telephone directory to asking a conversational information seeking agent to dial a specific person's office number. In academic terms, Information Retrieval can be formally defined as :

Definition. Information Retrieval(IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers) [2].

1.1.1 Document and Collection

A *document* is a file in text format for storing information on a storage media, especially for use by computers. A document consists of some minimal structures like title, author, date, subject, content etc. Documents can be in the form of web pages, emails, books, news articles, text messages, blogs etc.

A set of similar documents is called a *collection.* IR systems usually operate on a collection with documents stored in a pre-defined format (pdf, txt, doc etc.).

1.1.2 How Information Retrieval is different from tradional databases

Databases are datasets where related data is stored in a structured nature. Databases mainly consists of tables, where each record in a table can be uniquely identified with a special attribute called key. Each record in a database comprises of well defined attributes or fields. And each field in a record has a specific datatype. Therefore searching in a database is easy and efficient. Whereas on the other hand documents consists of free, unstructured texts. And there are no restrictions on the content of a document. As there is no defined structure, searching something from a large collection of documents becomes a hard problem. The main focus of IR is to address this difficulty to improve the efficiency and accuracy of the search results.

Structured queries can be used for comparison with the well defined semantics of database records which makes searching in a database easier. The following is an example of a structured query :

```
select name from Employee where salary > 50,000
```

Here Employee is a table in a company's database where name and salary are its attributes and this query returns the names of the employees whose salary is greater than fifty thousand.

Similar to documents, the queries in IR are also unstructured, hence a user need not be aware of any semantics or structured query language before posing a query. One can query in the naural language. An example of a query in IR is as follows :

```
Average paid employees
```

Due to the boom of the Big-Data scenario, and reduction in the cost of storage devices, people now have access to terabytes data available in unstructured nature. Storing such huge volumes of data in a well defined structured way is very difficult and time consuming, so IR is gaining popularity these days, as it is user friendly and can handle unstructed data efficiently.

1.2 The Retrival process

The retrieval procedure mainly comprises of two components or sub-processes : *Indexing* and *Retrieval*. Figure 1.1 shows the overview of a Retrieval system.

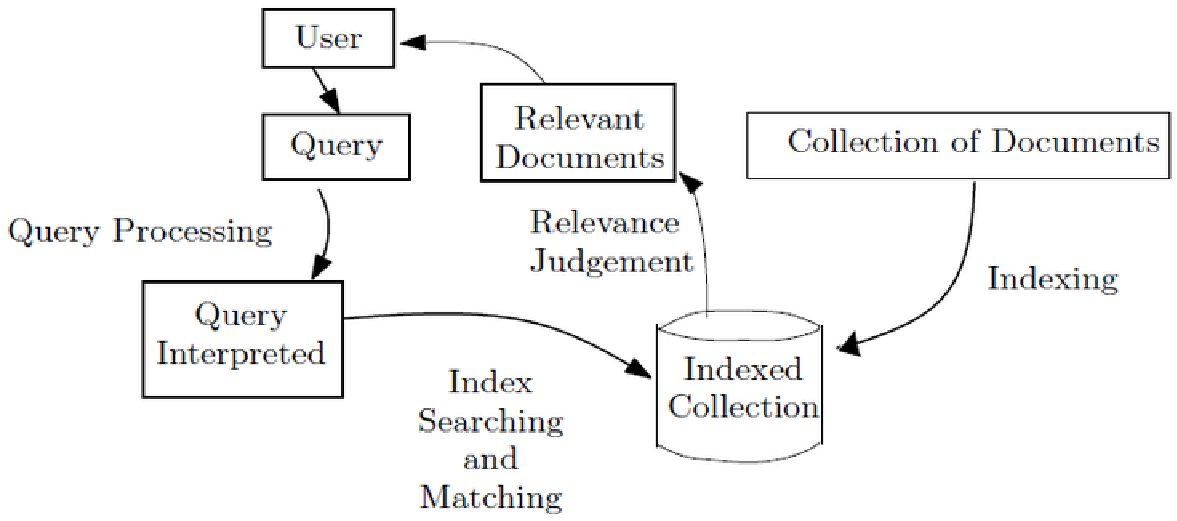

Figure 1.1: Overview of an Information Retrieval system [4].

1.2.1 Indexing

Indexing is the process by which documents are converted to data structures that enable faster search, i.e. precomputing as much as possible. Indexing consits of the following steps :

1. Reading and parsing the document.
2. Stopword Rmoval.
3. Stemming the words in the document.
4. Inserting each term in the data structure(index).

Definition. Stop Words are words that are very common such as *"the"*, *"and"*, thus are assumed to carry very little standalone meaning for searching, since nearly every document will contain the word, so it fails to discriminate between the relevant and non relevant documents and thus can be removed.

Definition. Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word, it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root [3].

Efficiency of an IR system is largely determined by the data structure it uses to store the indexed documents. Therefore choice and design of proper data structures is very important. One of the dominating indexing data structure for supporting basic search algorithms is *Inverted Index.*

For a term t, the Inverted Index data structure stores a list of IDs of all the documents that contain t. The term set is then organized in a suitable data structure for example an array, a hash table, a binary search tree etc. Figure 1.2 shows an example of the Inverted Index data structure.

1.2.2 Retrieval

The Retrieval process can only be performed after the documents have been indexed. The user presents his information need in the form of a query. The query is then parsed and stemmed using the same parser and stemmer that have been used during

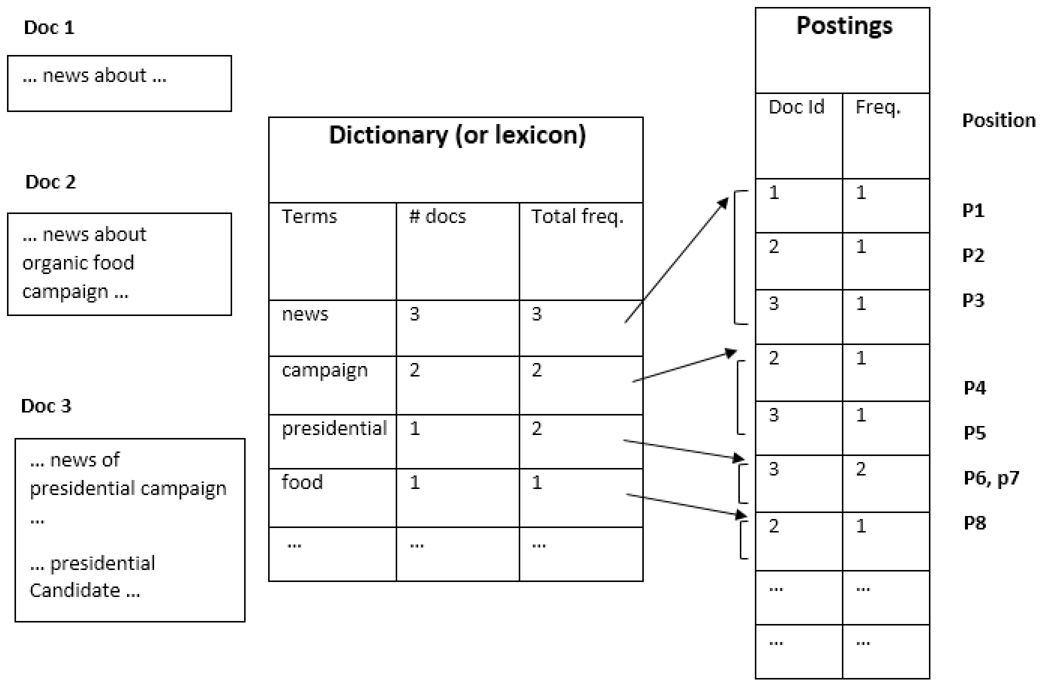

Figure 1.2: An example of Inverted Index data structure.

indexing. Sometimes additional operations like *Query Expansion* is also performed as a query processing step.

After processing the query, the query terms are matched with the index terms, if a match is found, the documents containing the matched term are treated as relevant documents. Then the matched documents are ranked according to some ranking function and the set of ranked documents are presented to the user as a response to his information need.

In some cases the subset of the ranked set of documents, that have been examined and judged as relevant by the user are treated as feedback documents. These feedback documents are used to initiate a user feedback cycle. In such a cycle, the system uses the documents selected by the user to change the query formulation. The modified query is assumed to be a better representation of the user's real need. This process is known as *Relevance Feedback*.

1.3 Evaluation of an IR system

Evaluating retrieval results is a key issue for IR systems. One common assumption is that the retrieval result is presented as a ranked list of documents. In binary relevance judgement, documents are classified into two categories : relevant and non relevant. The ranking positions of the relevant and non relevant documents are the major concern for most evaluation metrics.

As large document collections are used for retrieval evaluation these days, it is not affordable to judge all the documents retrieved any more. Therefore, incomplete relevance judgment, or partial relevance judgment, is commonly used. A pooling policy is used, i.e. for a group of runs submitted to a task for a given topic, only a certain number (say, 100) of top-ranked documents from all or some submitted runs are put into a pool. All the documents in the pool are judged, and all those documents that are not in the pool are not judged and are treated as irrelevant documents.

1.3.1 Basic requirements for the evaluation of an IR system

The performance evaluation of an ad-hoc IR system requires the following three components in a test collection :

1. A document collection
2. A test set of queries
3. A set of relevance judgement, which is a binary judgement (relevant or non relevant) for a query-document pair.

The document collection and the test set of queries should be of reasonable size (a minimum of 50 queries is standard).

In recent years, there are quite a few IR evaluation test collections such as TREC [i], NCTIR [ii], CELF [iii] and FIRE [iv]. A document in the test collection is judged as relevant or non relevant according to the user's need. This classification is referred to as the *ground truth* of relevance judgement.

[i] `http://trec.nist.gov/`
[ii] `http://research.nii.ac.jp/ntcir/index-en.html`
[iii] `http://www.clef-initiative.eu/`
[iv] `http://www.isical.ac.in/~clia/`

1.3.2 Evaluation metrics

In binary relevance judgement, many different metrics such as average precision, recall-level precision, precision at k (10 is most commonly used, other common options include 5, 20, and 100) document level, normalized discounted cumulative gain, the reciprocal rank, and many others, have been proposed for retrieval evaluation. Most of these commonly used metrics are ranking-based.

Average Precision

Let C be a document collection, q be the given query, and $total\_r$ be the total number of relevant documents in C for q. Let an IR system(irs) return a ranked list of documents $L = < d_1, d_2, ,,, ,d_n >$. Average precision(AP) over all relevant dcouments is defined as :

$$AP = \frac{1}{total\_r} \sum_{i=1}^{total\_r} \frac{i}{t_i} \tag{1.1}$$

where t_i is the ranking position of the i-th relevant document in the resultant list L.

If all $total\_r$ relevant documents does not appear in L, then the following equation can be used instead (m is the number of relevant documents that appear in L) :

$$AP = \frac{1}{total\_r} \sum_{i=1}^{m} \frac{i}{t_i} \tag{1.2}$$

where *nu(i)* is a function, which gives the number of relevant documents in the top i documents. *napd_best* is a normalization coefficient, which is the best possible NAPD value for such a resultant list.

In recent years, *Mean Average Precision*(MAP) has become a standard parameter. It has been observed that MAP has good discrimination and stability among evaluation metrics. Let Q be the query set, MAP of Q is the average of AP_{q_i} $\forall$ $q_i \in Q$. Hence :

$$MAP = \frac{1}{\mid Q \mid} \sum_{i=1}^{|Q|} AP_{q_i} \tag{1.3}$$

Recall-level Precision

Recall-level precision is defined as the ratio of the number of relevant documents retrieved in $L(num\_ret\_r)$ to the total number of relevant documents$(total_r)$. So :

$$Recall = \frac{num\_ret\_r}{total\_r} \tag{1.4}$$

Precision at k document level

Precision at 10 document level(P@k) is defined as the percentage of relevant documents in the top k documents in L.

Normalized Discounted Cumulative Gain

In Normalized Discounted Cumulative Gain(NDCG@k) each ranking position in a resultant document list is assigned a given weight. The top ranked documents are assigned the heaviest weights since they are the most convenient ones for the users to read. The first k documents are assigned a weight of *1*; then for any document ranked i that is greater than k, its weight is *w(i) = ln(k)/ln(i)*. For a resultant list of upto m documents, its *discounted cumulative gain*(DCG@k) is defined as :

$$DCG = \sum_{i=1}^{m} (w(i) * r(i)) \tag{1.5}$$

where *r(i)* is defined as: if the i-th document is relevant, then *r(i)* = 1; if the i-th document is non relevant, then *r(i)* = 0. DCG@k can be normalized using a normalization coefficient *DCG_best*, which is the DCG value of the best resultant list. Therefore :

$$NDCG = \frac{1}{DCG\_best} \sum_{i=1}^{m} (w(i) * r(i)) = \frac{DCG}{DCG\_best} \tag{1.6}$$

bpref

Incomplete relevance judgement's pooling policy does not affect some metrics such as precision at a given cut-off document level. However, in the evaluation of information retrieval systems, both precision and recall are important aspects and many metrics concern both of them at the same time. The pooling method is reliable, but recall is overestimated since it is likely that 30% - 50% of the relevant documents are not found. Therefore, some alternative metrics have been defined for incomplete relevance judgment. For a topic with *total_r* relevant documents where r is a relevant document and n is a member of the first *total_r* judged non-relevant documents as retrieved by the system, *bpref* is defined as :

$$bpref = \frac{1}{total\_r} \sum_{r} 1 - \frac{\mid n \ ranked \ higher \ than \ r \mid}{total\_r} \tag{1.7}$$

1.4 Language models for Retrieval

A common suggestion to users for coming up with good queries is to think of words that would likely appear in a relevant document, and to use those words as the query. The language modeling approach to IR directly models this idea: a document is a good match to a query if the document model is likely to generate the query, which will in turn happen if the document contains the query words often. Instead of overtly modeling the probability $P(R = 1 \mid q, d)$ of relevance of a document d to a query q, as in the traditional probabilistic approach to IR, the basic language modeling approach instead builds a probabilistic language model θ from each document d, and ranks documents based on the probability of the model generating the query: $P(q \mid \theta)$ [2].

Let the query q be a n term query, then q can be written as $q = q_1 q_2 \cdots q_n$. A unigram language model is assumed to generate text by generating each word independently. Thus, $p(q) = p(q_1 q_2 \cdots q_n) = p(q_1)p(q_2) \cdots p(q_n)$. Therefore the maximum likelihood estimator of the unigram model is given by :

$$p(q_i \mid \theta) = p(q_i \mid d) = \frac{tf(q_i,\ d)}{\mid d \mid} \tag{1.8}$$

Definition. To compute a score between a query term t and a document d, based on the weight of t in d. The simplest approach is to assign the weight to be equal to the number of occurrences of term t in document d. This weighting scheme is referred to as *term frequency* and is denoted by *tf*.

1.4.1 Smoothing

The key intuition behind smoothing is that if a non occuring term in the document occurs in the query q then the probability $p(q) = p(q_1 q_2 \cdots q_n) = p(q_1)p(q_2) \cdots p(q_n)$ will become zero and the term(s) in the query which also occurs in the document will be penalized and as a result the document will be scored low, to solve this issue smoothing is required.

The maximum likelihood unigram language model based on term frequencies in the collection(C) as a whole is given by :

$$p(q_i \mid \theta_C) = p(q_i \mid C) = \frac{tf_C(q_i,\ C)}{\mid C \mid} \tag{1.9}$$

Jelinek-Mercer Smoothing

Jelinek-Mercer smoothing creates a mixture model with both the distributions (document model(θ) and collection model(θ_C)), i.e. it mixes the probability from the document with the general collection frequency of the word. It is given by :

$$\begin{aligned} p(q_i \mid d, \theta_C) &= \lambda \cdot p(q_i \mid \theta) + (1-\lambda) \cdot p(q_i \mid \theta_C) \\ \Rightarrow p(q_i \mid d, \theta_C) &= \lambda \cdot \frac{tf(q_i,\ d)}{\mid d \mid} + (1-\lambda) \cdot \frac{tf_C(q_i,\ C)}{\mid C \mid} \end{aligned} \tag{1.10}$$

High value of λ is more conjunctive and tends to retrieve documents containing all query words. Low value of λ is more disjuctive and is suitable for long queries. Thus, correctly setting λ is very important for good performance.

Chapter 2

Word Embedding using Word2Vec

The word2vec model by Mikolov et al. and its applications have attracted a great amount of attention in recent years. The vector representations of words learned by word2vec models have been shown to carry semantic meanings and are useful in various IR and Natural Language Processing(NLP) tasks.

2.1 Continuous Bag-of-Words model of Word2Vec

2.1.1 Single-word context

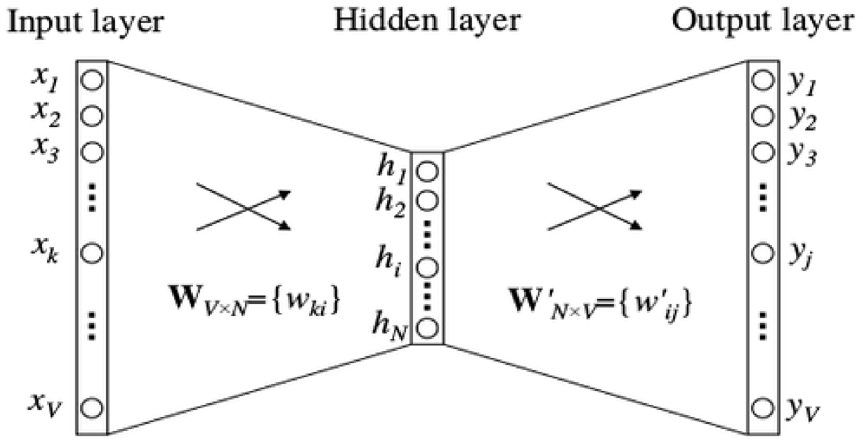

Figure 2.1: A simple CBOW model with one word context [5]

The continuous bag-of-words model (CBOW) was introduced by Mikolov et al. in 2013. In the most simple version of this model one word is considered per context, which means that the model will predict one target word given one context word. Figure 2.1 shows the network architecture under this simplified context definition.

In this setting V is the vocabulary size and N is the size of the hidden layer. The adjacent layers are fully connected. The input to the network is a one-hot encoded vector i.e. for a given context word only one out of V, $\{x_1, \cdots, x_V\}$, units will be *1* and all other units will be *0*.

The weights between the input and the hidden layer can be represented by a $V \times N$ matrix W. Each row of W is a n-dimensional representation v_w of the word presented at the input layer. For a given context word, ($x_k = 1$ and $x_{k'} = 0 \; \forall \; k' \neq k$) :

$$h = W^T x = v_{w_I}^T \tag{2.1}$$

i.e. the k-th row of W is copied to h, where v_{w_I} it the vector representation of the input word w_I. This implies that the activation function of the hidden unit directly passes its weighted sum of inputs to the next layer.

The weight matrix between the hidden layer and the ouput layer is W'. It is a $N \times V$ dimensional matrix. These weights are used to calculate a score u_j for each word in the vocabulary,

$$u_j = {v'_{w_j}}^T h \tag{2.2}$$

where v'_{w_j} is the j-th column of W'. Then a softmax classifier is used to obtain the posterior probability of the words :

$$p(w_j \mid w_I) = y_j = \frac{exp(u_j)}{\sum_{j'}^{V} exp(u_{j'})}$$

$$\Rightarrow p(w_j \mid w_I) = \frac{exp({v'_{w_j}}^T v_{w_I})}{\sum_{j'}^{V} exp({v'_{w_{j'}}}^T v_{w_I})} \tag{2.3}$$

v_w and v'_w are both representations of the word w. v_w comes from the rows of the input to hidden weight matrix W and v'_w comes from the hidden to output weight matrix W'.

The training objective (for one training sample) is to maximize 2.3, the conditional probability of observing the actual output word w_O given the input context word w_I with regard to the weights is :

$$\begin{aligned} max(p(w_o) \mid p(w_I)) &= max(y_{j^*}) \\ &= max(\log(y_{j^*})) \\ &= u_{j^*} - \log(\sum_{j'=1}^{V} exp(u_{j'})) = -E \end{aligned} \tag{2.4}$$

where $E = -\log(p(w_o) \mid p(w_I))$ is the loss function, and j^* is the index of the actual output word in the output layer.

2.1.2 Multi-word context

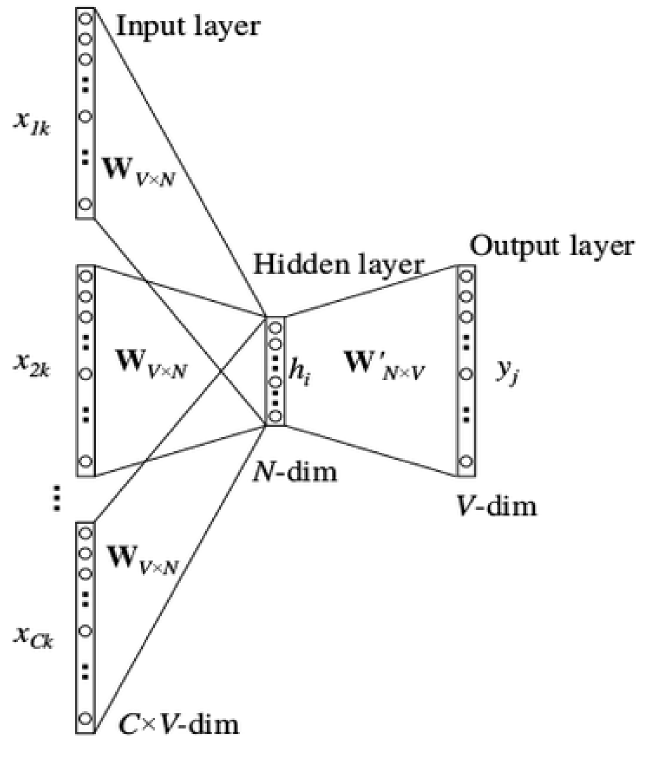

Figure 2.2: CBOW model with multiple words context [5]

Figure 2.2 shows the CBOW model with a context of multiple words. The hidden layer of this CBOW model takes the average of the vectors of the input context words as input, and uses the product of the input to hidden weight matrix and the average vector as the output.

$$\begin{aligned} h &= \frac{1}{C}W^T(x_1 + x_2 + \cdots + x_C) \\ &= \frac{1}{C}(v_{w_1} + v_{w_2} + \cdots + v_{w_C})^T \end{aligned} \tag{2.5}$$

where C is the number of words in the context, w_1, $\cdots$, w_C are the words in the context and v_w is the input vector of a word w. The loss function is :

$$\begin{aligned} E &= -\log(p(w_O \mid w_{I,1}, \cdots, w_{I,C})) \\ &= -u_{j^*} + \log(\sum_{j'=1}^{V} exp(u_{j'})) \\ &= -{v'_{w_O}}^T \cdot h + \log(\sum_{j'=1}^{V} exp({v'_{w_{j'}}}^T \cdot h)) \end{aligned} \tag{2.6}$$

which is same as the objective of the single-word context model, except that h is different, as defined in 2.4.

The update equation for the hidden to output layer weights is :

$$ {v'_{w_j}}^{(new)} = {v'_{w_j}}^{(old)} - \eta \cdot e_j \cdot h \qquad for\ j = 1, 2, \cdots, V. \tag{2.7}$$

where e_j is the prediction error of the output layer and $\eta > 0$ is the learning rate. The above equation needs to be applied to every element of the hidden to output layer weight matrix for each training instance.

For updating the weights of the input to hidden layer the following equation is to be applied to every word $w_{I,c}$ in the context.

$$v_{w_{I,c}}^{(new)} = v_{w_{I,c}}^{(old)} - \frac{1}{C} \cdot \eta \cdot EH^T \qquad for\ c = 1, 2, \cdots, C. \tag{2.8}$$

where *EH*, a N-dim vector, is the sum of the output vectors of all words in the vocabulary, weighted by their prediction error and $v_{w_{I,c}}$ is the input vector of the c-th word in the input context.

2.2 Computational Optimization

The CBOW model has two representstions for each word in the vocabulay : the input vector v_w and the output vector v'_w. Learning the input vectors is cheap; but learning the output vectors is very expensive. From the update equation 2.7, in order to update v'_w, for each training instance, one has to iterate every word w_j in the vocabulary, compute their net input u_j, probability prediction y_j, their prediction error e_j and finally use their prediction error to update their output vector v'_j.

Performing such computations for all the words, for every training instance is very expensive, making it impractical to scale up to large vocabularies or large training corpora. To solve this problem, an intuition is to limit the number of output vectors that must be updated per training instance. An elegant approach is perform sampling.

2.2.1 Negative Sampling

The idea of negative sampling is more straightforward : in order to deal with the difficulty of having too many output vectors that need to be updated per iteration, one only needs to update a sample of them.

The ground truth output word, i.e. the positive sample should be present in the sample so that its corresponding weights get updated and a few other negetive words should also be sampled so that the networks learns to discriminate between the correct and the erroneous words. A probabilistic distribution is needed for the sampling process, and it can be arbitrarily chosen. This distribution is called the noise distribution, and it is denoted as $P_n(w)$. word2vec uses a unigram distribution raised to the $\frac{3}{4}$-th power for the best quality of results.

In word2vec, instead of using a form of negative sampling that produces a well-defined posterior multinomial distribution, the authors argue that the following simplified training objective is capable of producing high-quality word embeddings :

$$E = -\log(\sigma({v'_{w_O}}^T h)) - \sum_{w_j \in \omega_{neg}} \log(\sigma(-{v'_{w_j}}^T h)) \tag{2.9}$$

where w_O is the output word, i.e. the positive sample, and v'_{w_O} is its output vector. h is the output value of the hidden layer, in CBOW :

$$h = \frac{1}{C} \sum_{c=1}^{C} v_{w_c} \tag{2.10}$$

$\omega_{neg} = \{w_j \mid j = 1, \cdots, K\}$ is the set of words that are sampled based on $P_n(w)$, i.e. the negative samples.

The update equations of the word vectors under negative sampling is given by :

$$v_{w_j}^{'\,(new)} = v_{w_j}^{'\,(old)} - \eta(\sigma(v_{w_j}^{'\,T} h) - t_j) \tag{2.11}$$

where t_j is the "label" of word w_j . $t = \mathit{1}$ when w_j is a positive sample, $t = \mathit{0}$ otherwise. The output vector only needs to be applied to $w_j \in \{ w_O \} \cup \omega_{neg}$ instead of every word in the vocabulary.

Chapter 3

Related Work

A document may not explicitly contain the terms present in the query. Still the document may be relevant with respect to the idea of information need presented by the query. If a relevant document does not contain the terms that are in the query, then that document will not be retrieved. The aim of *query expansion* is to reduce this query-document mismatch by expanding the query using words or phrases with a similar meaning or some other statistical relation to the set of relevant documents [4].

3.1 Using Word Embedding for Automatic Query Expansion

IR and Neural Network have started exploring deep learning based techniques for various IR problems in recent years. Word Embeddings generated using NNs is the focus of many recent IR researchers. Word Embeddings as discussed in the previous chapter gives a vector representation of the words in the corpus. If a and b are two words, and $\vec{a}$ and $\vec{b}$ are their embeddings, then it is expected that the distance between $\vec{a}$ and $\vec{b}$ is a quantitative indication of the semantic relatedness between a and b. The semantic relatedness between words is generally accurately captured by the vector similarity between the corresponding embeddings produced by Word2Vec. Thus, it provides a convenient way of finding words that are semantically related to any given word. As the purpose of query expansion is to find the words that are semantically similar to a given query, word embeddings obtained from Word2Vec can be used to select suitable terms that will improve the effectiveness of automatic query expansion.

In [6] the authors have designed the following methods for automatic query expansion:

1. Pre-retrieval kNN (Nearest Neighbour) based approach
2. Post-retrieval kNN based approach

3.1.1 Pre-retrieval kNN based approach

For a given query Q ($\{q_1 \cdots q_m\}$), C the set of candidate expansion terms has been defined as :

$$C = \cup_{q \in Q} \ NN(q) \tag{3.1}$$

where *NN(q)* is the set of k terms that are closest to q in the *embedding space*. For each candidate expansion term t in C, the mean *cosine similarity* between t and all the terms in Q have been computed according to the following equation :

$$Sim(t, Q) = \frac{1}{\mid Q \mid} \sum_{q_i \in Q} t \cdot q_i \tag{3.2}$$

The terms in C have been sorted on the basis of the above mean score and the top K' terms has been selected as the actual expansion terms.

3.1.2 Post-retrieval kNN based approach

In this approach, the authors have used a set of pseudo-relevant documents (PRD) — documents that were retrieved at top ranks in response to the initial query — to restrict the search domain for the candidate expansion terms. Instead of searching for nearest neighbours within the entire vocabulary of the document collection, only those terms that occured within PRD were considered. The size of PRD have been varied as a parameter. The remaining procedure for obtaining the expanded query was the same as section 3.1.1.

3.1.3 Extended Query Term Set

When the nearest neighbours of individual query words are considered for choosing expansion terms it might not reflect the information need properly as individual query words might have many senses but when combined together they might convey a different meaning altogther, the authors have captured this by composing the vectors of two adjacent terms (bigrams) in the query.

For a given query Q consisting of m terms $\{q_1 \cdots q_m\}$, Q_c, the set of query words bigrams was constructed as :

$$Q_c = \{\langle q_1, q_2\rangle, \langle q_2, q_3\rangle, \cdots, \langle q_{m-1}, q_m\rangle\} \tag{3.3}$$

The embedding for a bigram $\langle q_i, q_{i+1}\rangle$ was obtained simply by adding $\vec{q_i}$ and $\vec{q_{i+1}}$, where $\vec{q_i}$ and $\vec{q_{i+1}}$ are the embeddings of the words q_i and q_{i+1}. An extended query term set $Q^{'}$ has been defined as :

$$Q^{'} = Q \cup Q_c \tag{3.4}$$

In the above approaches $Q^{'}$ has been considered instead of Q to integrate the effect of compositionality.

3.1.4 Retrieval

For retrieval, the language model with Jelinek Mercer smoothing has been used. The following query model for the expanded query has been used :

$$p(w \mid Q_{exp}) = \alpha\, p(w \mid Q) + (1-\alpha)\, \frac{Sim(w,Q)}{\sum_{w' \in Q_{exp}} Sim(w',Q)} \tag{3.5}$$

where Q_{exp} is the set of top $K^{'}$ terms from C, the set of candidate expansion terms. As discussed in section 3.1.3 $Q^{'}$ can be used instead of Q. α is the interpolation parameter that has been used as the likelihood estimate of a term in the query, in combination with the normalized vector similarity with the query.

3.2 Query Expansion with Locally-Trained Word Embeddings

Corpus statistics often makes sense absent in some information. But often times the analysis that is made, is topically constrained. For example one might be analysing the *sports* documents in the collection. The language in such a domain is specialized and the distribution of over the word(w)-context(c) pair is unlikely to be similar to $p_c(w,c)$, where p_c is the distribution of the word-context pairs in the training corpus and can be estimated from corpus statistics. Prior works in IR also suggests that documents on subtopics in a collection have very different unigram distributions compared to the whole corpus.

In general a corpus consists of sufficiently diverse data. The support of $p_t(w, c)$ (the probability of observing a word-context pair conditioned on the topic t) is much smaller than and contained in that of $p_c(w, c)$. The loss, l, of a context that occurs more frequently in the topic, will be amplified by the importance weight :

$$\omega = \frac{p_t(w, c)}{p_c(w, c)} \tag{3.6}$$

As topics require specialized language, it is likely that training with the whole corpus will underemphasize these contexts.

The highly weighted terms may have a low value for $p_t(w, c)$ but a very high value relative to the corpus. The weights can be adjusted by considering the pointwise Kullback-Leibler divergence for each word w :

$$D_w(p_t \mid\mid p_c) = p_t(w) \log \frac{p_t(w)}{p_c(w)} \tag{3.7}$$

Words which have a much higher value of $p_t(w)$ than $p_c(w)$ and have a high absolute value of $p_t(w)$ will have high pointwise KL divergence.The higher ranked terms (i.e. good query expansion candidates) tend to have much higher probabilities than found in $p_c(w)$. If the loss on those words is large, it might result in poor embeddings for the most important words in the topic.

The authors of [7] has observed a dramatic change in distribution between the corpus and the topic that has implications for performance precisely because of the objective used by word2vec. The training emphasizes on word-context pair that occur with high frequency in the corpus. Even with heuristic downsampling of frequent terms in word2vec, these techniques resulted in inferior performance for specific topics. A qualitative difference was observed in $p_c(w, c)$ and $p_t(w, c)$ by training two word2vec models: the first on the large, generic Gigaword corpus and the second on a topically-constrained subset of the gigaword.

3.2.1 Local Word Embeddings

In information retrieval scenarios users rarely provide the system with examples of topic-specific documents, instead they provide a small set of keywords. IR techniques have been used to generate a query-specific set of topical documents. A language modeling approach have been adopted for doing so. In a language model, each document is represented as a maximum likelihood language model estimated from document term frequencies. Query language models are estimated similarly, using term frequency in the query. A document score then, is the Kullback-Leibler divergence between the query and document language model, which is given by :

$$D(p_q \mid\mid p_d) = \sum_{w \in \nu} p_q(w) \log \frac{p_q(w)}{p_d(w)} \tag{3.8}$$

Documents whose language models are more similar to the query language model have a lower KL divergence score.

The scores in equation 3.8 have been passed through a softmax function to derive a multinomial over the entire corpus :

$$p(d) = \frac{exp(-D(p_q \mid\mid p_d))}{\sum_{d'} exp(-D(p_q \mid\mid p_{d'}))} \tag{3.9}$$

The query-based multinomial, *p(d)*, provides a weighting function capturing the documents relevant to the topic. Although an estimation of the topic-specific documents from a query is imprecise (i.e. some nonrelevant documents might be scored highly), the language use tends to be consistent with that found in the known relevant documents.

A local word embedding have been trained using an arbitrary optimization method by sampling documents from *p(d)* instead of the entire corpus.

3.2.2 Query Expansion with Word Embeddings

As language model has been used for retrieval, query expansion involves the estimation of an alternative to p_q. As each expansion term is associated with a weight, the weights have been normalized to derive the expansion language model, p_{q^+}. This language model has then been interpolated with the original query model as :

$$p_q^1(w) = \lambda \, p_q(w) + (1 - \lambda) \, p_{q^+}(w) \tag{3.10}$$

where λ is the interpolation parameter that has been used as the likelihood estimate of a term in the query in combination with the expansion language model. The interpolated language model have been referred to as the expanded query score of a document.

The weight of an expansion term is equal to $UU^T q$, where U is a $\nu \times k$ term embedding matrix and q is a $\nu \times 1$ column term vector for a query. The top k terms have been taken and their weights have been normalized to compute p_{q^+}.

3.3 Adaptive Relevance Feedback in Information Retrieval

In existing embedding based query expansion methods the balance, between the original query language model p_q and the expansion language model p_{q^+}, is usually controlled by some parameter(α), which is often set to a fixed value across all the queries and collections. However, due to the variations of queries and expansion terms, this balance parameter presumably should be optimized for each query and each set of expansion terms. One needs to carefully balance the original query and the expansion terms because if the expansion terms are over trusted, the query might drift, but under-trusting it would not take advantage of query expansion, therefore having a perfect balance is very critical for efficient performance.

In relevance feedback also has a parameter that balances the original query and the feedback information. The authors in [8] have studied this novel problem and have proposed an adaptive relevance feedback method to dynamically predict an optimal balance coefficient using machine learning.

The authors have proposed the following heuristics to characterize feedback coefficients:

1. Discrimination of query: One expects that more discriminative the query is, the more drifting-tolerant it could be, and thus it would be safe to utilize more feedback information.
2. Discrimination of feedback : It can be hypothesized that clearer feedback documents could be trusted more.

Following the above three heuristics, the authors have explored a number of features and have combined them using logistic regression to predict the feedback coefficient.

3.3.1 Discrimination of Query

More discriminative a query is, the more drift tolerant it is, thus it is safe to utilize more feedback information. Therefore it is expected that the discrimination of query is correlated with the feedback coefficient. The following measures have been proposed to quantify it :

1. **Query Length :**

   A longer query is generally more discriminative than a short one, so the first feature selected by the authors is query length, which is formally defined as:

$$\mid Q \mid = \sum_{w \in Q} c(w, Q) \tag{3.11}$$

   where *c(w, Q)* is the count of term w in Q.

2. **Entropy of Query :**

   The queries are often very short, so the query entropy score is computed based on top-N resultant documents $(F^{'})$ of the initial retrieval, which is defined as :

$$QEnt\_A = \sum_{w \in F'} -p(w \mid \theta_{F'}) \log_2 p(w \mid \theta_{F'}) \tag{3.12}$$

   where $p(w \mid \theta_{F'})$ is estimated as $p(w \mid \theta_{F'}) = \frac{c(w,F^{'})}{\sum_w c(w,F^{'})}$.

3. **Clarity of Query**

   The clarity of a query is the Kullback-Leibler divergence of the query model from the collection model.

   Query clarity computation requires the estimation of the query language model, which, however, involves an interpolation between the original query model θ_Q and the pseudo feedback model $\theta_{F'}$, as well as setting an interpolation coefficient. To avoid such problems the authors did not estimate an entropy for the interpolated query model instead they have computed two separate clarity scores based on θ_Q and $\theta_{F'}$ and have used them directly as features in the supervised learning framework, leaving the optimization of their combination to the training process. The relative entropys are defined as :

$$QEnt\_R1 = \sum_{w \in Q} p(w \mid \theta_Q) \ \log \frac{p(w \mid \theta_Q)}{p(w \mid C)} \tag{3.13}$$

$$QEnt\_R2 = \sum_{w \in F'} p(w \mid \theta_{F'}) \ \log \frac{p(w \mid \theta_{F'})}{p(w \mid C)} \tag{3.14}$$

where $p(w \mid C)$ is the collection language model.

3.3.2 Discrimination of Feedback Documents

The documents, that are judged relevant by the user, have only been used for feedback and no negative feedback documents have been considered. Therefore if feedback documents are more discriminative it means that they focus more on the relevant topic and far from noise, thus discriminative feedback documents have been trusted.

1. **Feedback Length :**

   The number of feedback documents, i.e. feedback length $\mid F \mid$, has been used as one of the features. It is defined as :

$$\mid F \mid = \sum_{d} \delta(d, F) \tag{3.15}$$

   where $\delta(d, F) = 1$ if document $d \in F$, otherwise 0.

2. **Feedback Radius :**

   Feedback radius can be used to measure the broadness of the feedback documents, i.e. whether the feedback documents are concentrated on similar topic or not. Feedback radius can be defined as the average divergence between each document and the centroid of the feedback documents, which can be approximated using the Jensen-Shannon divergence among feedback document models. It is defined as :

$$FBRadius = \frac{1}{\mid F \mid} \sum_{d \in F} \sum_{w \in d} p(w \mid \theta_d) \log \frac{p(w \mid \theta_d)}{p(w \mid \theta_{centroid})} \tag{3.16}$$

   where $p(w \mid \theta_{centroid}) = \frac{1}{|F|} \sum_{d \in F} p(w \mid \theta_d)$.

3. **Entropy of Feedback Documents :**

   Feedback length and feedback radius, captures the discrimination of feedback documents on the document level, whereas the entropy of feedback documents, measures the term distribution, on the term level. Similarly to the computation of query entropy, the entropy of feedback document model θ_F is defined as :

$$FBEnt\_A = \sum_{w \in F} -p(w \mid \theta_F) \log_2 p(w \mid \theta_F) \tag{3.17}$$

where $p(w \mid \theta_F)$ is estimated as $p(w \mid \theta_F) = \frac{c(w,F)}{\sum_w c(w,F)}$.

4. **Clarity of Feedback Documents :**

   Similar to query clarity *QEnt_R1*, the feedback entropy *FBEnt_R1* is computed as :

$$FBEnt\_R1 = \sum_{w \in F} p(w \mid \theta_F) \ \log \frac{p(w \mid \theta_F)}{p(w \mid C)} \tag{3.18}$$

3.3.3 Learning Algorithm

For any relevance feedback model the balance parameter $\alpha \in [0, 1]$, therefore the authors have chosen logistic regression for learning the values of $\hat{\alpha}$ for different queries, using the above heuristically defined features as input, as logistic regression can take as input any value from -∞ to ∞, whereas the output is confined to values between 0 and 1.

Logistic regression models are of the form :

$$f(z) = \frac{1}{1 + exp(-z)} \tag{3.19}$$

where variable z represents some set of features, while *f(z)* represents the probability of a particular outcome, given that set of features. The authors have used *f(z)* as the predicted value for the coefficient α, and has interpreted it as the probability that one would use only the feedback model (as opposed to the original query model) in the mixture model formed by interpolating the two models. Variable z is a measure of the total contribution of all the features used in the model, defined as $z = \bar{w}\bar{x}$. $\bar{x}$ is a vector of numeric values representing the features. And $\bar{w}$ represents a set of weights, which indicates the relative weights for each feature. A positive weight means that the corresponding feature increases the probability of the outcome, while a negative weight means that its corresponding feature decreases the probability of that outcome, a large weight means that the feature strongly influences the probability of that outcome, while a near-zero weight means that the feature has little influence on the probability of that outcome.

Chapter 4

Proposed Methods

4.1 Approach 1 : Automatic Query Expansion using Pseudo-local Embeddings

4.1.1 Motivation

Query expansion intuitively requires terms that are semantically similar to query terms. We have seen in the previous chapter that the word embeddings obtained from Word2vec capture the semantic relationship between words very well. Thus, we chose to use word embeddings for selecting suitable terms for query expansion.

In [7], we the authors have shown that globally-trained word embeddings may not work well for queries where the query terms have more than one sense, as global embeddings only tend to capture the globally dominant sense of such words, so in this chapter we propose a novel approach that we refer to as pseudo-local embedding based query expansion. The local embedding approach in [7] gives good accuracy but it does so at the expense of a slower retrieval rate, as the word embeddings are trained on the fly during retrieval (on the resultant documents obtained during the initial retrieval), so we prefered a *pseudo-local* embedding for expanding the queries over local embedding.

4.1.2 Grouping similar topics by Clustering

It has been observed in previous works [6] [7] that the related word's vectors (obtained from Word2Vec) remain close to each other in the embedding space.

It has been observed that on training word embeddings on a topically constrained corpus, the nearest neighbours of embeddings yield better expansion terms, which are more related to the topic [7].

We also know that clustering algorithms aim to maximize the inter-cluster distance and minimize the intra cluster distance i.e. the clustering algorithms focus on grouping similar data together.

Keeping all the above observations in mind we have used the *k-means* clustering algorithm, with random seed initialization, to model the topics in the corpus, where each cluster(ς_i) is expected to represent a *topic set*. Word2Vec [i] has been trained on the entire collection to obtain the word embeddings. 1000 [ii] clusters have been formed, using the *k-means++* [iii] software package in java, these are assumed to represent 1000 different topics in the collection.

4.1.3 Selection of Core Terms

Definition. When a set of related documents are grouped together, the terms that represent the topic of discussion in the set of documents are called *core terms*. In our case the terms that represent the main matter of discussion in a cluster are called *core terms*.

We know that the *centroid* of a cluster is a point in the space that captures the central information about the cluster. The points that are close to the cluster center also contain central information about the cluster (similar to the centroid). As the points (terms in our case) that are close to the centroid form the core of the cluster, they are the most *noise-free* terms of the cluster. Therefore, we have selected $k(= 30)$ [iv] nearest neighbours of the cluster centoid as *core terms*(C_{j,ς_i}) that represent the topic of the cluster. The nearest neighbours were selected based on the cosine similarity between the globally embedded word vectors.

[i] Google's implementaion of Word2Vec has been used : `https://code.google.com/p/word2vec/`

[ii] We have arbitrarily chosen this parameter, it has not been varied as varying it is very much compute expensive.

[iii] `https://en.wikipedia.org/wiki/K-means%2B%2B`

[iv] The parameter k has been varied in the set $\{30, 40\}$ but $k = 30$ worked better, other variations have not been tried as the remaining procedure, which is very much time and space expensive, depends on this parameter.

4.1.4 Initial Retrieval and Cluster Pruning

Each individual core term, C_{j,ς_i}, of cluster ς_i was treated as a query and was used for retrieving documents from the entire collection. The union of resultant documents (top *thousand* documents) over all j core terms, per cluster, was taken to form a pseudo-topically-constrained corpus. Language model with Jelinek-Mercer smoothing was used for retrieval (with parameter 0.5 [v]).

The above method has been performed for all ς_i clusters. The clusters whose core term set retrieved less than *three thousand* documents, in total, were pruned, as such clusters represent very rare topics or random noise. The final number of clusters obtained after the pruning procedure was *644*.

4.1.5 Learning Pseudo-local Embeddings

We have learned the pseudo-local word embeddings for each cluster, ς_i, by training Word2Vec on the pseudo-topically-constrained corpus obtained per cluster. These word embeddings are assumed to carry the senses of the words that are dominant in the topically constrained sub-corpus.

Note : all the above procedure have been done at index time, no training of word embeddings is required during retrieval.

4.1.6 The final Retrieval Process with pseudo-locally expanded queries

For expanding a given user query, we first need to select the pseudo-local embedding from which the expansion terms are to be chosen. We have chosen $k^{'}(=3)$ [vi] closest core terms per query term, based on cosine similarity of their globally embedded word vectors. These $k^{'}$ nearest core terms ($C_{j,q},\ j \in \{1,2,\cdots,k^{'}\}\ ,\ q \in Q$) were used to select the corresponding pseudo-local embeddings (trained on the documents retrieved by the corresponding core term set to which the chosen core term belongs) for selecting expanding terms per query term.

[v] this parameter has not been varied as we have kept this parameter fixed throught all experiments, both in the case of baselines and proposed methods, so it won't affect the comparison.

[vi] $k^{'}$ has been varied in $\{3,5\}$, but $k^{'}=3$ worked better.

$k''(= 30)$ [vii] most similar terms ($t_{l,j,q},\ l \in \{1, 2, \cdots, k''\}$) to the core term, $C_{j,q}$, were chosen from the corresponding pseudo-local embedding $E_{j,q}$. The net similarity of a similar term, $t_{l,j,q}$, is given by :

$$Net\_sim_{t_{l,j,q}} = Sim(q, C_{j,q}) \cdot Sim(C_{j,q}, t_{l,j,q}) \tag{4.1}$$

where $Sim(q, C_{j,q})$ is the cosine similarity between the query word, q ($q \in Q$), and the core term, $C_{j,q}$, and $Sim(C_{j,q}, t_{l,j,q})$ is the cosine similarity between the core term, $C_{j,q}$ and the similar term, $t_{l,j,q}$. The set of net similarities of the similar terms corresponding to a core term have been *sum to one* normalized.

Union of all such similar words ($\cup_{q \in Q} \cup_{j \in \{1,2,cdots,k'\}} t_{l,j,q}$), were taken per core term per query. From this union, ξ most similar terms (U_ξ represents the top ξ terms in the union), based on their net similarity, were chosen as the final expansion terms for the query Q. The Query model, $\theta_{Q_{exp}}$, for the expanded query, Q_{exp} is given by :

$$p(w \mid Q_{exp}) = \alpha\, p(w \mid Q) + (1 - \alpha) \frac{Net\_sim_{t_{l,j,q}}}{\sum_{t' \in U_\xi} Net\_sim_{t'_{l,j,q}}} \tag{4.2}$$

where α is the interpolation parameter used as balance between the original query model and the expansion terms.

4.1.7 Experimental Setup

We have performed our experiments on TREC disk 4 and 5 collection and have used the query sets of TREC 6 (50 *queries*), 7 (50 *queries*), 8 (50 *queries*) and TREC *robust* (100 *queries*) for evaluating the performance of our retrieval system. Parameter setting has been done on the query set, TREC 6 and the remaining query sets have been used as test query sets. The parameter α has been varied in $\{0.1, 0.2, \cdots, 0.9\}$ and the number of expansion terms has been varied in the range $\{10, 20, \cdots, 120\}$.

For training the word embeddings, we have used an embedding size of 400 and a window size of 5. Negative sampling with 5 negative samples have been used for training using the CBOW architecture. The model has been trained for 80 epochs.

Indexing has been done using the Lucene [viii] package in Java. During indexing, stop

[vii] the parameter k'' has been varied in $\{30, 40, 50\}$. All of them yielded the same result, so we stuck with $k'' = 30$

[viii] `https://lucene.apache.org/core/`

Query Set	Method	Parameters		Metrics			
		# Expansion Terms	α	MAP	P@10	NDCG@10	bpref
TREC 6	LM - JM	-	-	0.2346	0.3720	0.4098	0.2662
	Global (Pre-ret)	70	0.3	0.2379	0.3940	0.4319	0.2648
	Local	120	0.2	0.2442	0.3720	0.4213	0.2674
	Proposed pseudo-local	100	0.5	0.2382	0.3740	0.4194	0.2639
TREC 7	LM - JM	-	-	0.1779	0.3800	0.4005	0.1900
	Global (Pre-ret)	70	0.3	0.1902	0.3940	0.4229	0.2018
	Local	120	0.2	0.1934	0.3960	0.4135	0.2043
	Proposed pseudo-local	100	0.5	0.1861	0.3840	0.4070	0.1972
TREC 8	LM - JM	-	-	0.2441	0.4320	0.4332	0.2629
	Global (Pre-ret)	70	0.3	0.2603	0.4620	0.4796	0.2737
	Local	120	0.2	0.2641	0.4640	0.4852	0.2768
	Proposed pseudo-local	100	0.5	0.2609	0.4720	0.4847	0.2762
Robust	LM - JM	-	-	0.2678	0.3929	0.3741	0.2619
	Global (Pre-ret)	70	0.3	0.2863	0.4152	0.4022	0.2731
	Local	120	0.2	0.2945	0.4232	0.4068	0.2793
	Proposed pseudo-local	100	0.5	0.2838	0.4141	0.3922	0.2702

Table 4.1: Comparison of our proposed method with different baselines.

words have been removed using the SMART [ix] stopword list, and the words have been stemmed using Porter Stemmer.

4.1.8 Results and Discussion

Table 4.1 shows the comparison among the baseline language model with Jelinek-Mercer smoothing, Pre-Retrieval query expansion with global embeddings (Roy et al.) [6], query expansion with local embeddings (Diaz et al.) [7] and our proposed pseudo-local embedding based query expansion.

Mean Average Precision(MAP) is the most standard metric in IR, as it is stable and has good discriminative ability. Comparing the above results based on MAP we found that our proposed method has outperformed Roy et. al.'s query expansion from globally embedded terms in TREC 6 and TREC 8 query sets whereas the latter has outperformed the proposed method in TREC 7 and TREC *robust* query sets. The proposed method has performed better than the baseline language model with Jelinek-Mercer smoothing in all the query sets. But Diaz's query expansion with locally embedded word vectors has out performed all the methods on all the query sets. However the local embedding based method has a practical drawback, its retrieval operation is time expensive as it learns the word embeddings on the fly during retrieval.

[ix] `ftp://ftp.cs.cornell.edu/pub/smart/`

	TREC 6	TREC 7	TREC 8	Robust
Global Embedding (Pre-ret) Vs Local Embedding	×	×	×	×
Global Embedding (Pre-ret) Vs Proposed Pseudo-Local Embeddng	×	×	×	×
Local Embedding Vs Proposed Pseudo-Local Embeddng	×	×	×	×

× : can't be claimed as significantly different.
✓ : significantly different.

Table 4.2: T-test of the proposed method with the other methods.

The performance in terms of MAP, P@10, NDCG@10 and bpref is comparable in all the methods. So, we have done pair-wise significant tests between the methods. Table 4.2 shows the T-test results between the methods, on all the four query sets.

4.2 Approach 2 : Adaptive Query Expansion using Locally and Globally embedded word vectors

4.2.1 Motivation

On comparing the previous approach with the other word embedding based methods, we observed that Dias et al.'s method of query expansion with locally trained word embeddings worked better than all the other methods. Also, Roy et al.'s Pre-Retrieval query expansion with globally trained word embeddings worked better than the proposed method in TREC 7 and TREC *robust* query sets. So, we analysed these two methods on per query level and observed that the performance depends a lot on the balancing parameter and the number of expansion terms added. Figure 4.1 shows difference in AP, per query, between the two methods. From the figure it is evident that local embedding doesnot work well for all queries, global embedding works much better for almost half of the queries.

The above observations and Lv et al.'s work on adaptive relevance feedback [8] inspired us to design an oracle that will combine the two methods of Diaz et al. and Roy et al. by learning the balancing parameters α and β and the number of expansion terms for each method, through machine learning.

4.2.2 Problem Formulation

We represent the expansion language model of Diaz et. al. as p^{+}_{Local} and the expansion language model of Roy et. al. as p^{+}_{Global}. Let p_Q be the original query model, then the expanded query model can be represented as :

$$p_{Q_{Exp}}(w) = \alpha\, p_Q + (1-\alpha)\,(\beta\, p^{+}_{Global} + (1-\beta)\, p^{+}_{Local}) \tag{4.3}$$

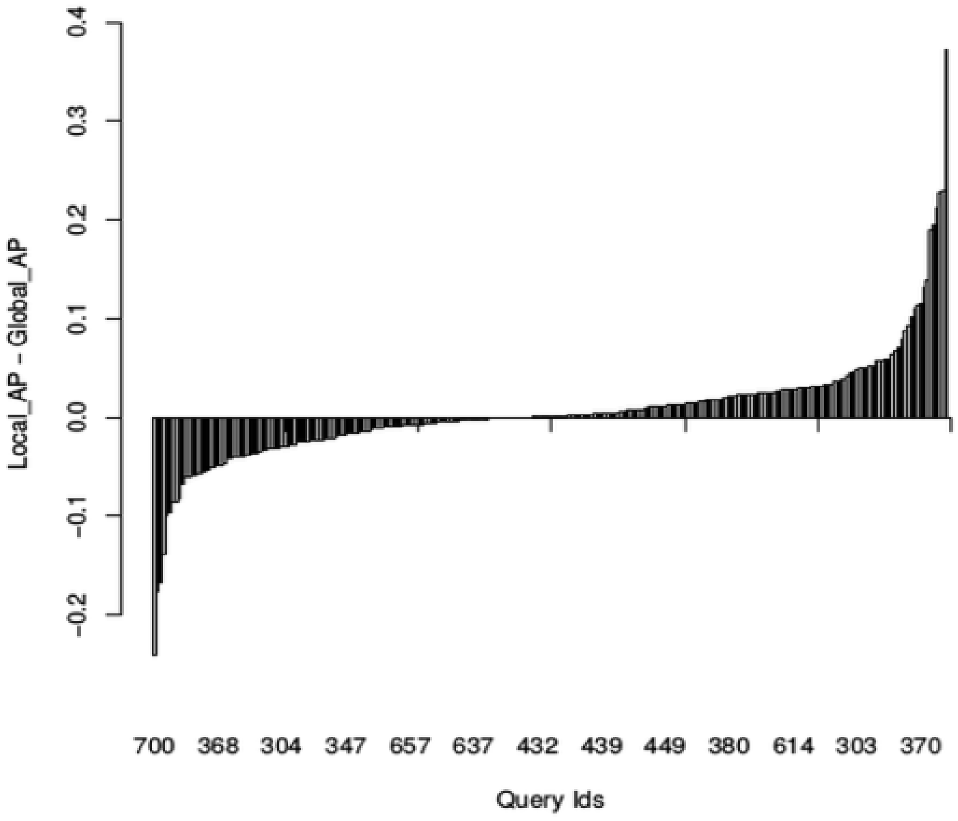

Figure 4.1: Plot of per query difference in AP between local embedding based query expansion of Diaz and Global Embedding based query expansion of Roy.

where α ($\in [0, 1]$) is the balancing parameter between the original query model and the expansion language models and β ($\in [0, 1]$) is the balancing parameter between the global expansion language model and the local expansion language model.

Our goal is to optimize the balancing parameters α and β for different queries. We tried to design functions, that can map a given query, Q to its appropiate balancing parameters α and β ($\alpha = F_1(Q)$ and $\beta = F_2(Q)$).

We also tried to design two more functions, that can map a given query, Q, to the number of global (Exp_{Global}) and local (Exp_{Local}) expansion terms it requires ($Exp_{Global} = F_3(Q)$ and $Exp_{Local} = F_4(Q)$).

4.2.3 Feature Selection

Inspired from [8] and [9] we have selected the following features as input to our machine learning algorithm that tried to learn the functions F_1, F_2, F_3 and F_4.

1. **Query Entropy :**

   As the queries are often very short, so we have computed the query entropy score based on the top-N resultant documents ($F^{'}$) of the initial retrieval, which is defined as :

$$Q_E = \sum_{t \in \theta_{F'}} -p(t \mid F^{'}) \log_2 p(t \mid \theta_{F'}) \tag{4.4}$$

   where $p(t \mid \theta_{F'})$ is estimated as $p(t \mid \theta_{F'}) = \frac{c(t,F^{'})}{\sum_t c(t,F^{'})}$, where *c(t, $F^{'}$)* is the count of term t in $F^{'}$.

2. **Simplified Query Clarity Score :**

   The simplified clarity score (SCS) measures the Kullback-Leibler divergence of the (simplified) query language model from the collection language model, as an indication for query specificity. The KL-divergence between the query Q and the collection C is computed as follows :

$$Q_{SCS} = \sum_{t \in Q} p(t \mid \theta_Q) \log_2 \frac{p(t \mid \theta_Q)}{p(t \mid C)} \tag{4.5}$$

SCS is strongly related to the *avgICTF* predictor, assuming each term appears only once in the query. In such a case, $Q_{Clarity} = \log \frac{1}{|Q|} + avgICTF(Q)$. Thus, SCS measures the specificity of the query while also taking into account the query length.

3. **Average Collection Query Similarity :**

   The vector-space based query similarity to the collection has been measured by considering the collection as a one large document composed of concatenation of all the documents. The collection query similarity (CQS) of a query term is defined as follows :

$$Q_{CQS}(t) = (1 + \log(tf(t, C))) \cdot idf(t) \tag{4.6}$$

   where $tf(t, C)$ is the *term frequency* of the term t in the collection C and $idf(t)$ is the *inverse document frequency* of the term t.

   Average collection query similarity is given by :

$$avgQ_{CQS} = \frac{1}{\mid Q \mid} \sum_{t \in Q} Q_{CQS}(t) \tag{4.7}$$

4. **Average variance of term weights over the documents containing it :**

   VAR($w(t, d)$) measures the variance of the term weights over the documents(d) containing it in the collection.The weight of a term that occurs in a document is determined by :

$$w(t, d) = (1 + \log(tf(t, d))) \cdot idf(t) \tag{4.8}$$

   Average variance is given by :

$$avgQ_V = \frac{1}{\mid Q \mid} \sum_{t \in Q} VAR(w(t, d)) \tag{4.9}$$

5. **Average pointwise mutual information :**

   The pointwise mutual information (PMI) is a popular measure of co-occurrence statistics of two terms in the collection. It is defined as :

$$PMI(t_1, t_2) = \log \frac{p(t_1, t_2 \mid C)}{p(t_1 \mid C)p(t_2 \mid C)} \tag{4.10}$$

where $p(t_1, t_2 \mid C)$ is the probability of the two terms to co-occur in the corpus, which can be approximated by maximum likelihood estimation.

Let B the set of all adjacent overlapping bigrams in the query Q. Then average pointwise mutual information is given by :

$$avgQ_{PMI} = \frac{1}{\mid B \mid} \sum_{(t_1,t_2) \in B} PMI(t_1, t_2) \tag{4.11}$$

6. **Pseudo-feedback Radius :**

   Pseudo-feedback radius (PFBR) can be used to measure the broadness of the pseudo feedback documents (documents retrieved by the query after initial retrieval), i.e. whether the pseudo feedback documents are concentrated on a single topic or not. Pseudo-feedback radius can be defined as the average divergence between each document and the centroid of the pseudo feedback documents, which can be approximated using the Jensen-Shannon divergence among pseudo feedback document models. It is defined as :

$$Q_{PFBR} = \frac{1}{\mid F' \mid} \sum_{d \in F'} \sum_{t \in d} p(t \mid \theta_d) \log \frac{p(t \mid \theta_d)}{p(t \mid \theta_{centroid})} \tag{4.12}$$

   where $p(t \mid \theta_{centroid}) = \frac{1}{|F'|} \sum_{d \in F'} p(t \mid \theta_d)$.

7. **Pseudo-feedback clarity score :**

   Similar to the simplified query clarity score, pseudo-feedback clarity score (PFBCS) can be calculated as :

$$Q_{PFBCS} = \sum_{t \in F'} p(t \mid \theta_{F'}) \; \log \frac{p(t \mid \theta_{F'})}{p(t \mid C)} \tag{4.13}$$

8. **Average Local/Global Term Similarity :**

   Term similarity, $Sim(t, q)$ is the cosine similarity between the term $t \in Nearest\_Neighbour(q)$ in the embedding space and the query term $q \in Q$.

Average local term similarity(Q_{ALTS}) is the average of $Sim(t, q)\ \forall\ t \in Nearest\_Neighbour(q)$ in the local embedding space, $\forall\ q \in Q$.

Average global term similarity(Q_{AGTS}) can be defined similarly.

9. **Range of Local/Global Term Similarity :**

Range of term similarities is defined as :

$$RTS = max(Sim(t, q)) - min(Sim(t, q)) \tag{4.14}$$

where $max(Sim(t, q))$ is the maximum term similarity $\forall\ t \in Nearest\_Neighbour(q)$ in the local embedding space, $\forall\ q \in Q$. Similarly $min(Sim(t, q))$ is the minimum term similarity.

Range of local term similarity is the RTS defined for the local embedding space, similarly range of global term similarity is the RTS defined for the global embedding space.

4.2.4 Learning Algorithm

The balance parameters α and β lie in $[0, 1]$, we have chosen logistic regression for learning the values of $\hat{\alpha}$ and $\hat{\beta}$ for different queries, using the above heuristically defined features as input, as in logistic regression the output is always confined to values between 0 and 1.

Logistic regression models are of the form :

$$F(z) = \frac{1}{1 + exp(-z)} \tag{4.15}$$

where the variable z represents the set of features. We have used $F_1(z)$ as the predicted value for α and $F_2(z)$ as the predicted value for β. Variable z is a measure of the total contribution of all the features used in the model, defined as $z = \bar{w}\bar{x}$. $\bar{x}$ is a vector of numeric values representing the features; and $\bar{w}$ represents a set of weights, which indicates the relative weights for each feature.

For predicting the number of global (Exp_{Global}) and local (Exp_{Local}) expansion terms, we have used linear regression as these values need not be confined between 0 and 1. For predicting the values of Exp_{Global} and Exp_{Local} we have not used the features, average local/global term similarity and range of local/global term similarity, as the computation of these features require Exp_{Global} and Exp_{Local}.

4.2.5 Experimental Setup

We have prepared the training set by performing a grid search on the parameters α ($\in \{0.0, 0.1, 0.2, \cdots, 0.9\}$), β ($\in \{0.0, 0.1, 0.2, \cdots, 0.9\}$), Exp_{Global} ($\in \{10, 20, \cdots, 120\}$) and Exp_{Local} ($\in \{10, 20, \cdots, 120\}$), and have selected the best parameter setting per query. All the features have been *zero-one* normalized. We have used the Python package, Tensorflow[x], by Google to implement the learning algorithm.

4.2.6 Discussion

The learning algorithm failed to learn the functions F_1, F_2, F_3 and F_4, possibly because of inadequate number of discriminative features or possibly because of inadequate amount of training samples (200 training samples from the query sets TREC $6, 7, 8$ and first 50 queries of TREC *robust*, the last 50 queries of TREC *robust* have been used as test samples). We have tried different optimizing techniques like Adam, Adagrad, stochastic gradient descent and RMS prop and have tried all possible parameter settings of these optimizers. We have even tried using dropouts in the weights $\bar{w}$, but nothing among this worked. The logistic regression and the linear regression just gave random outputs which were not even close to the target values.

The Classification Approach

We have also tried to map the problem as a classification problem, where given a query, Q, the classifier will choose whether to use Diaz et al.'s local embedding or Roy et al.'s global embedding for expanding the queries.

For this approach, we have prepared the ground-truths by comparing the query-wise eval files obtained by running TREC Eval[xi] on the resultant files of both the methods and have chosen the better performing method in terms of AP per query.

[x]`https://www.tensorflow.org/`

[xi]`https://trec.nist.gov/trec_eval/`

This method also failed to classify accurately, it classified all the test samples to the class of local embedding. We have obtained the following confusion matrix after running the classifier on the test set :

$$\begin{array}{ccc} & PredictedLocal & PredictedGlobal \\ TrueLocal & 30 & 0 \\ TrueGlobal & 20 & 0 \end{array} \tag{4.16}$$

4.3 Approach 3 : Combining Local and Global Embedding based Query Expansion methods by Data Fusion

4.3.1 Motivation

Data Fusion usually works when a highly ranked document in one retrieval system is lowly ranked in the other retrieval system and vice versa. On analysing the performance of local embedding based query expansion with pre-retrieval global embedding based query expansion we observed that almost half of the queries work better with local embedding and the remaining work better with global embedding. Thus, we decided to perform data fusion on these two methods.

4.3.2 Normalizing techniques

We have used the following normalizing techniques to normalize the scores that the resultant documents have received from both local and global embedding based query expansion methods :

1. **Zero-One normalization :**

   If there are m component results L_i, each of which is from an information retrieval system ir_i and contains n documents d_{ij} $(1 \leq i \leq m)$ and $(1 \leq j \leq n)$. r_{ij} is the raw score that d_{ij} obtains from ir_i. Then the scores of the documents in each L_i can be normalized in the range $[0, 1]$ using :

   $$s_{ij} = \frac{r_{ij} - min\_r_i}{max\_r_i - min\_r_i} \tag{4.17}$$

   where min_r_i is the minimal score that appears in L_i, max_r_i is the maximal score that appears in L_i, r_{ij} is the raw score of document d_{ij} , and s_{ij} is the normalized score for document d_{ij}.

2. **Sum-to-One normalization :**

   If there are a group of raw scores $\{r_1, r_2, \cdots, r_n\}$, from a ranked list of documents, then sum-to-one method normalizes raw scores using :

$$s_i = \frac{r_i}{\sum_{i=1}^{n} r_i} \tag{4.18}$$

4.3.3 Data Fusion Methods

Let there be a document collection C and a group of retrieval systems $IR = \{ir_i\}$ for $(1 \leq i \leq n)$. All retrieval systems ir_i $(1 \leq i \leq n)$ search C for a given query Q and each of them provides a ranked list of documents $L_i =< d_{i1}, d_{i2}, \cdots, d_{im} >$. A relevance score $s_i(d_{ij})$ is associated with each of the documents in the list. Data fusion techniques are use to merge these n ranked lists into one.

1. **CombSum :**

   The data fusion method *CombSum* generates the merged scores for every document, d, according to the equation :

$$g(d) = \sum_{i=1}^{n} s_i(d) \tag{4.19}$$

   where $s_i(d)$ is the score that ir_i assigns to d. If d does not appear in any L_i, then a default score (0) is assigned to it. $g(d)$ is the global score that every document d gets. After assignment of the global scores the documents are ranked according to their global score.

2. **CombMNZ :**

   The data fusion method *CombMNZ* assigns the global scores to the documents according to the equation :

$$g(d) = m * \sum_{i=1}^{n} s_i(d) \tag{4.20}$$

   where m is the number of results in which document d appears.

Query Set	Method	Parameters		Metrics			
		# Expansion Terms	α	MAP	P@10	NDCG@10	bpref
TREC 6	LM - JM	-	-	0.2346	0.3720	0.4098	0.2662
	Global(Pre-ret)	70	0.3	0.2379	0.3940	0.4319	0.2648
	Local	120	0.2	0.2442	0.3720	0.4213	0.2674
	Fusion method	-	-	0.2462	0.3860	0.4296	0.2682
TREC 7	LM - JM	-	-	0.1779	0.3800	0.4005	0.1900
	Global(Pre-ret)	70	0.3	0.1902	0.3940	0.4229	0.2018
	Local	120	0.2	0.1934	0.3960	0.4135	0.2043
	Fusion method	-	-	0.1926	0.4080	0.4260	0.2054
TREC 8	LM - JM	-	-	0.2441	0.4320	0.4332	0.2629
	Global(Pre-ret)	70	0.3	0.2603	0.4620	0.4796	0.2737
	Local	120	0.2	0.2641	0.4640	0.4852	0.2768
	Fusion method	-	-	0.2669	0.4840	0.4960	0.2797
Robust	LM - JM	-	-	0.2678	0.3929	0.3741	0.2619
	Global(Pre-ret)	70	0.3	0.2863	0.4152	0.4022	0.2731
	Local	120	0.2	0.2945	0.4232	0.4068	0.2793
	Fusion method	-	-	0.2949	0.4394	0.4170	0.2822

Table 4.3: Comparison of the fusion method with different baselines.

4.3.4 Results and Discussion

We have tried all the possible combinations of the normalizing techniques and data fusion methods described above, and have found that *CombSum* with *zero-one* normalization work equally well as *CombMNZ* with *zero-one* normalization. These two methods work better than the other possible combinations. As both these methods work equally well, we refer *CombSum* with *zero-one* as the final *fusion method*, without loss of generality.

Table 4.3 shows the comparison among the baseline language model with Jelinek-Mercer smoothing, Pre-Retrieval query expansion with global embeddings [6], query expansion with local embeddings [7] and our fusion method.

On comparing the Fusion method with the other methods, in terms of MAP, we found that the fused method has outperformed all the other methods in TREC 6, 8 and *robust* query sets, but Diaz et. al.'s local embedding based query expansion has beaten the Fusion method in TREC 7 query set. But on comparing the Fusion method with other methods in terms of the other metric like P@10, NDCG@10 and bpref, we find that the fusion method has outperformed all the other methods except for global embedding based query expansion, in terms of P@10, in TREC 6 query set. Thus we can come to the conclusion that fusing the two different embedding based query expansion techniques performs better than both of them according to our hyposthesis.

	TREC 6	TREC 7	TREC 8	Robust
Global Embedding (Pre-ret) Vs Local Embedding	×	×	×	×
Global Embedding (Pre-ret) Vs Fusion method	✓	×	✓	✓
Local Embedding Vs Fusion method	×	×	×	×

× : can't be claimed as significantly different.
✓ : significantly different.

Table 4.4: T-test of the Fusion method with the other methods.

The performance in terms of MAP, P@10, NDCG@10 and bpref is comparable in all the methods. So, we have done pair-wise significant tests between the methods. Table 4.4 shows the T-test results between the methods, on all the four query sets

4.4 An Observation

We have also compared the performance of Post-Retrieval query expansion with global embeddings of Roy et al. [6] and query expansion with locally trained word embeddings of (Diaz et al.) [7], by changing the parameter of Jelinek-Mercer smoothing and have found that the Post-retrieval method has performed better than the local embedding based method in terms of the metrics MAP, P@10, NDCG@10 and bpref for the parameter value 0.1 (This parameter value gave the best results over all possbile values for the parameter). Table 4.5 shows the comparison. The Post-Retrieval based method is also computaionally efficient in comparison with the Pre-Retrieval global embedding based method and local embedding based method, as it has to perform much less number of floating point operations compared to the Pre-Retrieval based method, as the vocabulary used here is much smaller than that of the Pre-Retrieval method and it also doesnot require any training during retrieval.

4.5 Conclusion

We have studied the use of term relatedness in the context of query expansion for *ad-hoc* information retrieval. We have proposed three approaches for improving the retrieval performance though word embedding based query expansion. In the first approach we have proposed a time efficient retrieval algorithm for query expansion using *pseudo-locally* constrained word embeddings. This method showed comparable performance with the global embedding based query expansion technique but the local embedding based expansion method was superior to our approach in all the cases. In our second approach we have tried to present a learning approach that adaptively predicts the balance co-efficients between the original query model and the local and global expansion language models. In this approach we have also tried to predict the

Query Set	Method	Parameters		Metrics			
		# Expansion Terms	α	MAP	P@10	NDCG@10	bpref
TREC 6	LM - JM	-	-	0.2339	0.3760	0.4126	0.2680
	Global(Post-ret)	100	0.5	0.2496	0.4020	0.4456	0.2719
	Local	120	0.3	0.2454	0.3780	0.4248	0.2710
TREC 7	LM - JM	-	-	0.1776	0.3920	0.4098	0.1884
	Global(Post-ret)	100	0.5	0.1990	0.4140	0.4473	0.2089
	Local	120	0.3	0.1916	0.4040	0.4200	0.2031
TREC 8	LM - JM	-	-	0.2453	0.4280	0.4270	0.2647
	Global(Post-ret)	100	0.5	0.2735	0.4680	0.4899	0.2781
	Local	120	0.3	0.2644	0.4600	0.4764	0.2796
Robust	LM - JM	-	-	0.2658	0.3909	0.3678	0.2590
	Global(Post-ret)	100	0.5	0.2959	0.4465	0.4209	0.2814
	Local	120	0.3	0.2949	0.4232	0.4036	0.2807

Table 4.5: Comparison between Post-Retrieval query expansion with global embeddings (Roy et al.) [6] and query expansion with locally trained word embeddings (Diaz et al.) [7].

optimal number of expansion terms required for the local and global embedding based query expansion methods. But this approach failed to perform in accordance to our hyposthesis. In our third approach we used data fusion techniques to fuse the results of query expansion based on local and global embeddings to have an improved performance over both the methods. The third approach could beat the local embedding based technique in three out of four query sets and it outperformed the global embedding based method is all the query sets, in term of *mean average precision* (MAP).

4.6 Future Work

There is still scope for improvement in the following areas :

1. In Approach 1, we are yet to try with other methods of topic modelling and selecting *core terms*. Other methods for choosing the relevant *pseudo-local embeddings* according to a user's query is yet to be explored.

2. In Approach 2, we are yet to figure out why the machine learning algorithms are failing to learn from the data, whether it is due to inadequate number of discriminative features or due to the lack in number of training samples or something else.

3. In Approach 3, there is room for trying out other linear and nor linear normalizing techniques. Fusion of Local and Global embeddings of different parameter

settings, other than the optimal settings, can also be tried to improve the results.

Bibliography

[1] Amit Singhal. Modern Information Retrieval: A Brief Overview. IEEE Data Eng. Bull., 24(4):35-43, Dec 2001.

[2] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. Introduction to Information Retrieval. Cambridge University Press, 2008.

[3] Wikipedia Contributors. Stemming. Wikipedia - The Free Encyclopedia, June 2018, `https://en.wikipedia.org/w/index.php?title=Stemming&oldid=843892309`.

[4] Parantapa Goswami. Query Expansion using WordNet. Indian Statistical Institute, Kolkata, Dissertation:2011-296, 2011, `http://hdl.handle.net/123456789/6453`.

[5] Rong, Xin. word2vec parameter learning explained. arXiv, preprint arXiv:1411.2738, Nov 2014.

[6] Dwaipayan Roy. Using Word Embedding for Automatic Query Expansion. arXiv, preprint arXiv:1606.07608, Jun 2016.

[7] Fernando Diaz. Query expansion with locally-trained word embeddings. arXiv, preprint arXiv:1605.07891, May 2016.

[8] Yuanhua Lv. Adaptive Relevance Feedback in Information Retrieval. Proceedings of the 18th ACM conference on Information and knowledge management, ACM, pp. 255-264, Nov 2009.

[9] David Carmel. Estimating the Query Difficulty for Information Retrieval. Morgan and Claypool publishers, 2010.

[10] Shengli Wu. Data Fusion in Information Retrieval. Springer, 2012.