Neural Query Expansion over Microblogs

Ismail Talha Yilmaz
Dept. of Computer Engineering
Hacettepe University
Ankara, Turkey
ismail_yilmaz@hacettepe.edu.tr

Engin Demir
Dept. of Computer Engineering
Hacettepe University
Ankara, Turkey
engindemir@cs.hacettepe.edu.tr

***Abstract*—The Internet has been widely used all around the world. People share lots of information about themselves or anything they experienced using the internet. Social media platforms such as microblogs are excessively used for that purpose. Twitter, which is one of the most popular platforms, is known to contain massive amounts of data. As the amount of data becomes larger, it becomes harder to retrieve useful tweets by the given information. One problem with traditional IR systems over microblogs like Twitter is, tweets are limited to 140 characters. This is a limiting factor for retrieving the results of queries. To fit into 140 words, people may drop what they aim to say or they may use informal abbreviations. This limits the information gained from those texts for IR tasks. Moreover, people can write anything they want without any structures through microblogs. Tweets are posted using informal language, which may cause missing or grammatically wrong sentences. This is also another problematic issue for IR systems that are focused on microblog texts. Besides those problems, the question is, can we improve the query retrieval accuracy over microblogs such as Twitter? In that context, one of the important IR techniques, Query Expansion, comes to help. To eliminate problems of microblogs, neural query expansion techniques can be used. In this work, it is proposed to apply neural query expansion over Twitter data. The main problem system aims to overcome, is to improve query efficiency over microblogs (Twitter) by proposing a new query expansion model, which makes this task crucial and makes this study different than the previous ones. The aim is to develop the recurrent neural network model such as LSTM and add the embedding similarity layer after it to detect which possible terms are good candidates for query expansion. Such a model is not applied to query expansion problems over microblogs in literature, which makes proposed work as an extension over previous models. Queries will be given to network and for each term in the query, the pipeline aims to retrieve the best terms suitable for expansion. Evaluations are done over the TREC 2011 Microblog Track dataset and the results show that the proposed methodology gives slightly better results than the baseline method.**

***Index Terms*—query expansion, encoder decoder models, word2vec, LSTM**

I. Introduction

In today's world, the internet has been widely used. People share and reach any information online. As a result, information becomes reachable without any limits. It is possible to learn what is happening from thousands of miles away at this instant. Not only general news produced by huge news centers but also personal data are presented from a wide variety of time range; from minutes to days. Microblogs such as Twitter contain millions of tweets posted each day. It produces huge personal information laying on the internet. This huge amount of information waiting to be discovered. As data is getting larger, it becomes harder to perform queries.

Query expansion is a technique in Information Retrieval. It aims to improve query performance by re-structuring queries. Using some techniques such as language models, relevance feedback, and word embeddings, new query terms are discovered and these terms are added to the original query. With the new form of the query, the aim is to minimize the query-document mismatch. This technique is very helpful in terms of search precision of the query. In this work, we aim to apply query expansion over TREC 2011 Microblogs dataset that contains Twitter data posted in 2011.

Twitter is one of the worst greatest social media platform that provides tweets sent by people all around the world. In this study, we will work on Twitter data. While making queries, there are some issues with microblogs that arise because of the nature of microblogs. Firstly, on Twitter, tweets are limited to 140 characters. To fit a post on Twitter, users sometimes have to drop what they intend to say, use fewer words or characters along with abbreviations that are informally defined. Although a reader can understand what he reads, character limitation becomes a problem for IR tasks. Such usages may limit information gained over text. Another issue with microblogs is, people can post anything they want without any format checking. Informal language usage may result in problems such as grammatically wrong sentences, missing words, repeating characters, or randomly added words. It is problematic for IR systems to process such textual data.

As we deal with textual data of microblogs, traditional techniques may not function as desired. For such situations, neural query expansion algorithms can be a nice choice. In this study, we propose a system, that achieves neural query expansion. Overall view of the system can be observed in "Fig. 1".

The contribution of this work is, we add embedding similarity method at the end of the neural network pipeline. The proposed system firstly fetches data from TREC in JSON format. It retrieves tweets from the JSON file and checks whether tweet language is English. It adds those English tweets to pickles. Then, the system makes preprocessing for all tweets. All tweets are lowercased and all punctuations are removed. Then special characters such as newline, tab, and quotation marks are removed. After that, we apply contraction

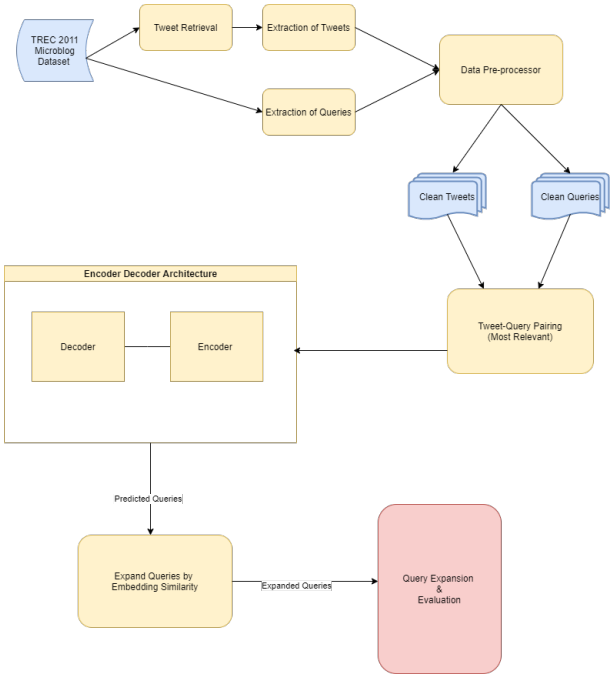

Fig. 1. Overall architecture of proposed system.

mapping to normalize all words in the same format. Finally, we remove English stopwords using an already created dictionary by NLTK. Then we tokenize all tweets word by word and apply post-padding to all words to have the same shape. As we don't have a suitable dataset for further steps, we built our dataset by matching tweets with the most relevant queries. BM25 algorithm is used for that purpose. Prepared data are fed into neural network architecture i.e. Seq2Seq network. This network contains an embedding layer, LSTMs in Encoder and Decoder parts. The result of the decoder are predictions and these predictions are fed into embedding similarity algorithm. Different numbers of most similar words to original queries are fetched and the results are added to the original query. Evaluations are done over the TREC 2011 Microblog Track dataset. Data contains 50 topics and these topics used as queries. Mean Average Precision (MAP) metric calculated as an evaluation metric. The results show that the proposed method with embedding similarity provides stable MAP scores.

The remaining of this paper is organized as follows. In section 2, some related works from the literature are discussed. In chapter 3, details of the proposed model are defined which contains dataset, data preprocessing, neural network, and embedding similarity parts. In chapter 4, experiments and their results are given. Finally in chapter 5 conclusion remarks are discussed.

II. Related Work

In the literature, various studies perform query expansion with different approaches.

The first paper is the work by Ltaifa et. al. that applies neural query expansion [1]. In this paper, the authors aim to apply query expansion over Twitter data using a hybrid neural network model. This model combines a feedforward neural network and a recurrent neural network. They generate word representations with fastText which takes character level n-grams. They proposed a hybrid-autoencoder based model for unsupervised feature selection and extraction. Finally, to improve the retrieval process effectiveness, a cluster-based microblog retrieval method is performed using the efficient features representations from the Hybrid Deep Neural Network structure. They made experiments over the TREC2011 Microblog dataset and experimental results show that the performance of clustering and especially information retrieval in microblogs depend heavily on features representation.

The second work is the work by Wang et. al. [2]. This work is not a neural query expansion, rather it is a classification based study. In that work, authors propose a k-Nearest Neighbor (kNN) based Query Expansion (QE) algorithm to generate words from local word embeddings for query expansion. Besides, they incorporate temporal evidences into the classification algorithm. Experiments are done over TREC Twitter corpora and results demonstrate that the proposed approach over performs the baseline approaches.

The third work in the literature is the work by Imani et. al. [3]. In that work, the authors implemented an embedding based query expansion method that includes an artificial neural network as a classifier. Siamese neural network is used to that aim. This neural network takes embeddings as input and predicts whether terms selected for query expansion are suitable or not. Terms are classified into three groups; good, bad, and neutral. Authors perform experiments on four TREC newswire and web collections. Results show that using terms selected by the classier for expansion significantly improves retrieval performance and shown to be more robust than baselines.

The final work about neural query expansion is the work by Zaiem and Sadat [4]. In that study, authors implement as Seq2seq model for query expansion. This study the first one that implements the Seq2seq network for query expansion. They built encoder-decoder architecture with pre-trained word embeddings, Glove. For evaluation, the IR part, they used Trec Robust 2004 dataset. Results of the experiment show that seq2seq based query expansion model shows good results but not very significant from baseline models. Our proposed work is an extension of that model.

III. Methodology

In this section, the methodology of the proposed system is discussed. Firstly, information about the dataset is given. Then data preprocessing, ranking query results, and neural network details along with embedding similarity calculation algorithm are discussed.

A. Dataset

To achieve neural query expansion task, we used TREC 2011 Microblogs dataset [4]. This is a dataset that contains tweets sampled in 2011. It contains 16 million tweets retrieved

between 23 January 2011 and 08 February 2011. The Tweets 2011 corpus is unusual such that data is not downloaded directly. Rather, a list of tweet identifiers is retrieved and the actual tweets are downloaded directly from Twitter using the open-source twitter tools project. However, to obtain the lists of tweets to be downloaded, a data usage agreement must be signed and after submitting, credentials are provided.

Downloaded data is in the JSON file. File contains several fields about tweets and only text part of the JSON should be retrieved. Another issue is tweets are not in a specific language, tweets from all over the world included. Hence we need to process each tweet, detect the ones that are written in English, and put those tweets into pickles. To do that, we use python's 'langdetect' library to retrieve relevant tweets that are in English.

It takes a huge amount of time and resources to retrieve 16 million tweets. Moreover, there are limitations with API to retrieve a limited number of tweets in a specified time. We could download tweets as long as the defined quota allows. Hence rather than retrieving all data, we only retrieved a subset of data, 10.000 tweets. The finalized file contains only those tweets. Data contains 50 topics that are defined as topics for those tweets. These topics will be used as queries for query expansion tasks.

This version of the dataset is not still suitable to use so we have to make some operations over the dataset to use in the project. Firstly, we made some preprocessing to clean the data. The aim is to clean and normalize the textual data. Preprocessed data are fed into the neural network and the results are sent to embedding similarity calculation algorithm pipeline. Details of pre-processing functions are explained in the next section.

B. Data Preprocessing

We work with the textual data. To feed text into the neural network pipeline, it has to be in the proper form. Hence, text data has to be preprocessed.

First of all as already mentioned, some special characters are removed from tweets to keep data stable. The same point with those characters is they don't give or change the meaning of the text. Rather than that, they brake the stability of text data for further processes. Those characters are carriage return character '/r', newline character '/n', tab character '/t' and quotation character ' " '. This process results in more smooth and stable text, removing unexpected space characters.

Secondly, all of the text in tweets are lowercased. The aim is to remove differences between uppercase and lowercase words. Moreover, by doing this, we can prevent some upper-lowercase typos.

Thirdly, some of the punctuation characters are removed. Those punctuation characters are hardcoded and fed to the system directly. These are question mark '?', colon ':', exclamation mark '!', dot '.', comma ',', and semicolon ';'. Just as special characters, punctuation characters also don't make sense for word meanings. However, we didn't eliminate single quotation marks here. The reason is quotation marks can also be used as possessive pronouns and we handle possessive pronouns separately.

Possessive pronouns are sentences to show belonging. They are sometimes added as ' 's ' suffix at the end of the word. If we directly eliminated quotation marks, the meaning of the sentence might change. For example, let's consider the word 'Hacettepe's'. If we eliminate the quotation mark, we could have 'Hacettepes' word and it becomes meaningless. Hence we eliminate 's completely. As a result of this operation, we have 'Hacettepe' word, which is the correct clean form and can be considered as the stem of that word.

Additionally, all of the words are lemmatized. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. We extract the lemmas of each word using an additional package; NLTK's WordNetLemmatizer. Wordnet is a large, freely, and publicly available lexical database for the English language aiming to establish structured semantic relationships between words. It offers lemmatization capabilities as well and is one of the earliest and most commonly used lemmatizers. NLTK offers an interface to it, but we have to download it first to use it.

We didn't do stemming, instead, we applied lemmatization. The difference between stemming and lemmatization is, lemmatization considers the context and converts the word to its meaningful base form, whereas stemming just removes the last few characters, often leading to incorrect meanings and spelling errors.

Another step is a contraction mapping. Contractions are shortcut techniques generally used in the English language. For example 'He'll' is written formally 'He will' or 'aren't' is a shortcut of 'are not'. What contraction mapping does is mapping shortcut forms i.e. contractions to formal versions. When working with textual data, contractions sometimes cause trouble and contraction mapping aims to resolve that issue. We didn't use a library for contraction mapping, we added contractions to the dictionary and used it for mapping.

Finally, stopwords are eliminated. A stopword is a commonly used word (such as the, a, an, in) that are meaningless by itself but to supportive words for other words. The language of the tweets in English, so we have to get English stopwords somewhere as it would be re-invention of the wheel by hard-coding all stopwords. To that aim, we used NLTK's stopword package. Again we have to download that package to retrieve stopwords correctly.

Neural network pipelines cannot process sentences effectively, so we have to split our data so that each part corresponds to a word. To achieve this, we applied tokenization and for each tweet, we split word by word. Over tokenized data, we apply post padding so that all of the data have the same shape.

So far all of the steps to pre-process and normalization of data is described. We feed this data to the neural network pipeline. However, one more step is needed to enter into a neural network, which will be described in the next section.

C. Ranking Query Results

So far, we fetched the data and normalized it. However, the current form of data is not capable of training a model over the encoder-decoder model as it is not labeled. Hence we have to prepare the dataset first. To achieve this, for each tweet in the corpus, we pick the most relevant query among the query list and make them pairs.

As the first step for preparing the dataset, queries are sampled from the pre-processed tweets. Retrieval is done using Okapi BM25 algorithm described in [5]. BM25 can be mathematically described as in formula "(1)"

$$IDF*((k+1)*TF)/(k*(1.0-b+b*(|d|/avgDl))+TF) \quad (1)$$

Analyzing the formula, tf is q's term frequency in the document d. IDF is the inverse document frequency weight of the query term q. —d— is the length of the document d in words and avgdl is the average document length in the text collection from which documents are drawn. k and b are free parameters, usually chosen, in the absence of an advanced optimization.

We need to process tweets along with the queries to retrieve the highest-ranked tweet to be matched with the query. To achieve this, data has to be tokenized. As we already did in the pre-processing step, we feed all of the data in tokenized form. For each query, we retrieve the highest-ranked tweet. We feed this data, query, and tweet pair, to neural network pipeline.

D. Neural Network Pipeline

For query expansion, we fed pre-processed tweets along with their topics to the neural network pipeline. As a neural network, we use seq2seq networks with encoder-decoder architecture.

1) Encoder Decoder Models: Encoder decoder models are widely used as Sequence-to-sequence (Seq2seq) models in sequence prediction problems and they have proven to produce stable results in areas like Machine Translation and Text Summarization. These models are generally composed of recurrent neural networks. Overview of the architecture is shown in "Fig. 2".

Encoder decoder models are developed by 2 RNNs. Encoders are responsible to encode the input text into vectors. Those vectors contain information of the entire input sequence and they are passed to the decoder. Decoders are responsible to decode those vectors into output representations that are generally text sequences. As already mentioned, encoders and decoders are composed of RNNs. But there is an issue with RNNs, especially in long texts. Although RNNs have feedback loops in the recurrent layer which allows them to keep information in 'memory' over time, it is problematic to train vanilla RNNs to solve problems that require learning long-term temporal dependencies. The reason is the gradient of the loss function vanishes exponentially with time. This is called a vanishing gradient problem. To overcome this issue,

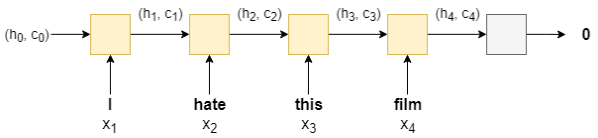

Fig. 2. Architecture of LSTM model.

Long Short Term Memory (LSTM) or Gated Recurrent Unit (GRU) networks can be used.

LSTMs are a type of RNN that uses special units inside that are not used in vanilla RNNs. Those units contain a 'memory cell' to keep the information in memory for long periods. Data comes in the memory and a combination of gates are used to control that data to input, output, and forgot. Example architecture of LSTM model is shown in "Fig. 2".

This architecture gives LSTMs the advantage to learn longer-term dependencies.

The formula of how LSTM model works presented in "(2)". h(i) means the hidden state and x is the word. c(i) is the LSTM specific part, which stands for the memory of the LSTM. To summarize, each layer takes its current word and the memory from the previous words to generate new hidden and memory states.

$$(h(i), c(i)) = LSTM(h(i-1), c(i-1), x) \quad (2)$$

GRUs are similar to LSTMs, but they are in a more simple structure. Like LSTMs, GRUs also use a set of gates to control the flow of data, but they don't use separate memory cells, and they use fewer gates. Hence, they give better performance compared to LSTMs but they don't produce as accurate results as LSTMs. Hence in our work, we used LSTMs in both encoder and decoder architectures.

As we work with textual data, the first step for entrance the network is an embedding layer, where embeddings are constructed from words. In this work, we set the dimension of the embedding layer to 200. That means we map words into 200 dimension vector representations.

Those vectors are passed firstly in the encoder part. As already mentioned, we used LSTMs in this proposed work. Encoder model contains three LSTMs connected. Each LSTM maps words to 300 dimensions called a latent dimension to detect latent features. The encoder model produces vector representation for all words in a sentence. This vector is passed to the decoder model to be decoded into output representations.

Likewise, the encoder model, decoder also contains an LSTM layer that contains 300 dimensions. In decoder, each LSTM unit accepts a hidden state from the previous unit and produces output as well as its hidden state. The result of LSTM is passed to the attention layer.

The problem with the traditional convolutional encoders is, they produce a single representation for the entire input sequence. When text size becomes long, this may create poor results. Here attention-based model comes into context. To avoid those issues, we added an attention layer that is proposed in work [8]

The result of the attention layer is concatenated with the result of LSTM and passed to a dense layer to create the final representation of calculated sequences. In the proposed model, we used softmax as an activation function. The result of the dense layer gives prediction for query expansion tasks.

The proposed model is trained with rmsprop optimizer. The loss function is a sparse categorical cross-entropy function. To avoid too much training, an early stopping function is used. The model trained with 200 epochs and the batch size selected is 256. The results of the model are written to a file to be processed on further parts in the pipeline.

So far model is trained. To work over real-world data that is unseen before, a slightly different process is applied called an inference layer. In this layer, firstly we encode the input sequence and return internal states. Then we start the decoder with setting the encoder internal states as the decoder's initial states. Next, we append the predicted result by the decoder to the decoded sequence. We repeat the process recursively and create final predictions. These predictions are passed to calculate embedding similarities module for further processing.

2) Embedding Similarity Module: In textual data, we represent words as vectors in multi-dimensional space. After mapping words to vectors, a need for computing similarity between vectors arise. The main principle here is, similar data points are close to each other on vectorial space. There are various ways to compute the similarity between vectors. Euclidean distance, Jaccard similarity [7] and Cosine similarity can be given as an example.

So far we have obtained predictions for possible queries from encoder-decoder structure. What we do in this module is, among predicted keywords, we select the most relevant ones with the tweet and add them to the original query. Hence we develop filtered expanded query as we filter among predictions and pick the relevant ones.

To achieve this, we use the pre-trained Google News embeddings. After representing words in vectorial space, we take each word of expanded query and measure cosine similarity between query word and tweet. We filter the most relevant tweets, i.e. the tweets that provide the highest similarity score. The number of related tweets to be filtered can be referred to as K. We made experiments with different K values and observe the results in Experiments & Results section.

Let's consider the case where we set the K value as 3. Here, we retrieve the 3 closest words among given predictions for a query. The results of this algorithm, which are 3 words, are added to the original query to create an expanded query. If prediction contains less than 3 words, then all of the words in prediction are taken. Moreover, among predicted words, the words that are also in the original query, are not added for expansion.

E. Query Expansion

After all of the steps explained in previous sections, we finally have the original query and expanded terms for the new query. Adding those terms to the original query, we construct an expanded query. For query expansion task, we use expanded query and retrieve the results.

For query retrieval, we implement the BM25 retrieval model. Details of the BM25 retrieval model are explained in the "Ranking Query Results" subsection. Using expanded queries, we retrieve results and perform evaluations over them.

In the next chapter, we explain the experiments and analyze their results.

IV. Experiments & Results

We perform some experiments for query expansion task. As an experiment, we compare the proposed model in this work with the model proposed in [4]. The baseline model implements the only sequence to sequence model for query expansion task. We added embedding similarity functionality and filter relevant keywords to the original query and we create a pipeline. In this experiment, we took the baseline model and applied it to our dataset. Then we apply our proposed model to the same dataset. For the evaluation of the proposed model, we used Mean Reciprocal Rank (MRR). Details of MRR are described in the next sub-section

A. Evaluation Metrics

Mean Reciprocal Rank (MRR) is simply mean of Reciprocal Rank (RR) at the point where the document is retrieved. For example, if the first rank retrieved document is relevant, MRR is 1. If at the second rank retrieved document is the first relevant MRR is 1/2. If at the third rank retrieved document is the first relevant one, then MRR is 1/3. This calculation goes on using that logic. The important thing in MRR is the rank of the first relevant document. To describe MRR mathematically, we can use the following formula.

$$\mathrm{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\mathrm{rank}_i}.$$

In this equation, rank(i) refers to the rank position of the first relevant document. Q value refers to the queries. The main aim is to calculate the mean of all RR scores.

B. Results

As the first experiment, we compare the proposed method with the baseline method. We refer to the baseline method as seq2seq and proposed model as seq2seq + embedding similarity. In this experiment, we set the default K value for embedding similarity to 3. That means, we retrieve 3 nearest neighbors of the predicted query to expanded query. As already explained, we calculated MRR scores for both methods and the results are presented at IV-B.

Method	MRR
seq2seq	0.259
seq2seq + embedding similarity	0.261

According to the results, although both models produce stable results, the proposed method results with slightly better scores on the MRR metric. However, we should also note that the differences are not so big.

As the second experiment, we compare different K values for the proposed method in calculating embedding similarity. K value stands for the number of nearest keywords filtered. The aim is to find an optimal K for our model. Recall that if K is higher than predicted word counts, all words in the query are taken. We calculate MRR scores for each K values in the query expansion task. Below the table, results are presented.

K-value	MRR
2	0.257
3	0.261
5	0.259
7	0.213

Experimental results show that the best MRR score is found when K is 3. K value of 3 gives a better score than 2. The reason is probably the K value of 3 is more informative for query expansion than the value of 2. Although K value increases until value 3, it starts to decrease after that value. The reason could be adding more terms to query may produce irrelevant documents to have a higher ranking.

V. Discussions

The proposed model uses seq2seq architecture. LSTMs are used in the encoder-decoder model. As already discussed, LSTMs contain hidden memory cells that are capable of containing past data. Keeping more gates may result in more calculations. In practice, the training encoder-decoder architecture of the proposed model takes a considerable amount of time. To overcome this, GRUs can be considered. GRUs don't contain that memory cell. This may produce unsatisfactory results however making fewer calculations probably boost the system performance. The pros and cons of using LSTMs and GRUs can be considered while developing encoder-decoder architectures. In this work, we sacrificed from performance and choose LSTMs.

In this study, we used TREC 2011 Microblog dataset that contains 16 million tweets posted between some time intervals in 2011. Dataset also contains a topics list, which consists of 50 different topics. However, it is a reality 16 million tweets can not fit into 50 categories. We paired the most relevant tweets with queries for each query. However, we suffer from that in our model. We observe that the proposed encoder-decoder model tends to categorize the tweets that are not part of 50 topics, to the last topic in the list. Hence, most of the tweets - which are not in 50 topics - are categorized under the same query although they are not relevant. Unfortunately, this produces irrelevant results. This can be the reason that produced MRR scores are generally low. In query expansion tasks, it can be more productive to use datasets that contain one most relevant query for each document in the corpus. In our case, the distribution of 10.000 tweets into 50 topics is not feasible. Using different, specific datasets for that approach, may improve evaluation scores.

VI. Conclusion

Query expansion is a critical task, especially in cases when we deal with microblog data. It is hard to expand queries effectively on microblogs as they have character limitations and have no pre-defined format. Neural query expansion is proposed to overcome those difficulties. To improve query performance over Twitter data, we propose a new method that also bases on neural query expansion.

In this work, we build a system that retrieves English texts among all tweets and applies different variations of pre-processing and normalization such as lemmatization and stopword removal over those tweets. As an extension over previous works in the literature, we build a neural network pipeline, that contains an encoder-decoder architecture along with embedding similarity calculator at the end of it. The results of this classifier are added to the original query to apply the query expansion task.

We work with TREC Microblogs 2011 dataset, which contains tweets posted in a specified time interval in 2011. As there is no pre-defined dataset for query expansion, we use tweet topics as queries and retrieve relevant tweets using the BM25 retrieval algorithm. According to the experiments, our model produces better results than previous work that only implements the seq2seq model in both MAP and MRR metrics. Moreover, we made experiments for different K values of the K-NN classifier and we obtained the best results when K is set to 5.

As a future work, seq2seq architecture in the proposed model can be improved to implement the pointer-generator model as described in [6]. This model may improve predictions made and create better-expanded queries. We already discussed the performance problem of seq2seq in "Discussion". As another future work, rather than using LSTMs, we plan to use GRUs in the architecture of the seq2seq model. Moreover, we plan to use a more suitable dataset that contains specific queries for each text in the corpus. This way, results can be improved.

References

[1] Ltaifa, Ibtihel Ben, Lobna Hlaoua, and Lotfi Ben Romdhane. "Hybrid Deep Neural Network-Based Text Representation Model to Improve Microblog Retrieval." Cybernetics and Systems 51.2 (2020): 115-139.

[2] Wang, Yashen, Heyan Huang, and Chong Feng. "Query expansion with local conceptual word embeddings in microblog retrieval." IEEE Transactions on Knowledge and Data Engineering (2019).

[3] Imani, Ayyoob, et al. "Deep neural networks for query expansion using word embeddings." European Conference on Information Retrieval. Springer, Cham, 2019.

[4] Zaiem, Salah, and Fatiha Sadat. "Sequence to Sequence Learning for Query Expansion." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 33. 2019.

[5] Trotman, Andrew, Antti Puurula, and Blake Burgess. "Improvements to BM25 and language models examined." Proceedings of the 2014 Australasian Document Computing Symposium. 2014.

[6] See, Abigail, Peter J. Liu, and Christopher D. Manning. "Get to the point: Summarization with pointer-generator networks." arXiv preprint arXiv:1704.04368 (2017).

[7] Niwattanakul, Suphakit, et al. "Using of Jaccard coefficient for keywords similarity." Proceedings of the international multiconference of engineers and computer scientists. Vol. 1. No. 6. 2013.

[8] Rush, Alexander M., Sumit Chopra, and Jason Weston. "A neural attention model for abstractive sentence summarization." arXiv preprint arXiv:1509.00685 (2015).