Task-Oriented Query Reformulation with Reinforcement Learning

Rodrigo Nogueira
Tandon School of Engineering
New York University
rodrigonogueira@nyu.edu

Kyunghyun Cho
Courant Institute of Mathematical Sciences
Center for Data Science
New York University
kyunghyun.cho@nyu.edu

Abstract

Search engines play an important role in our everyday lives by assisting us in finding the information we need. When we input a complex query, however, results are often far from satisfactory. In this work, we introduce a query reformulation system based on a neural network that rewrites a query to maximize the number of relevant documents returned. We train this neural network with reinforcement learning. The actions correspond to selecting terms to build a reformulated query, and the reward is the document recall. We evaluate our approach on three datasets against strong baselines and show a relative improvement of 5-20% in terms of recall. Furthermore, we present a simple method to estimate a conservative upper-bound performance of a model in a particular environment and verify that there is still large room for improvements.

1 Introduction

Search engines help us find what we need among the vast array of available data. When we request some information using a long or inexact description of it, these systems, however, often fail to deliver relevant items. In this case, what typically follows is an iterative process in which we try to express our need differently in the hope that the system will return what we want. This is a major issue in information retrieval. For instance, Huang and Efthimiadis (2009) estimate that 28-52% of all the web queries are modifications of previous ones.

To a certain extent, this problem occurs because search engines rely on matching words in the query with words in relevant documents, to perform retrieval. If there is a mismatch between them, a relevant document may be missed.

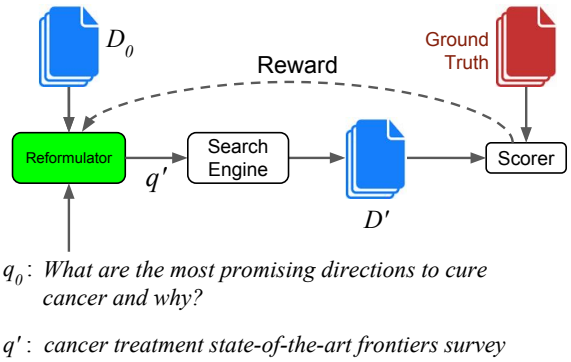

Figure 1: A graphical illustration of the proposed framework for query reformulation. A set of documents D_0 is retrieved from a search engine using the initial query q_0. Our reformulator selects terms from q_0 and D_0 to produce a reformulated query q' which is then sent to the search engine. Documents D' are returned, and a reward is computed against the set of ground-truth documents. The reformulator is trained with reinforcement learning to produce a query, or a series of queries, to maximize the expected return.

One way to address this problem is to automatically rewrite a query so that it becomes more likely to retrieve relevant documents. This technique is known as *automatic query reformulation*. It typically expands the original query by adding terms from, for instance, dictionaries of synonyms such as WordNet (Miller, 1995), or from the initial set of retrieved documents (Xu and Croft, 1996). This latter type of reformulation is known as pseudo (or blind) relevance feedback (PRF), in which the relevance of each term of the retrieved documents is automatically inferred.

The proposed method is built on top of PRF but differs from previous works as we frame the query

reformulation problem as a reinforcement learning (RL) problem. An initial query is the natural language expression of the desired goal, and an agent (i.e. reformulator) *learns* to reformulate an initial query to maximize the expected return (i.e. retrieval performance) through actions (i.e. selecting terms for a new query). The environment is a search engine which produces a new state (i.e. retrieved documents). Our framework is illustrated in Fig. 1.

The most important implication of this framework is that a search engine is treated as a *black box* that an agent learns to use in order to retrieve more relevant items. This opens the possibility of training an agent to use a search engine for a task other than the one it was originally intended for. To support this claim, we evaluate our agent on the task of question answering (Q&A), citation recommendation, and passage/snippet retrieval.

As for training data, we use two publicly available datasets (TREC-CAR and Jeopardy) and introduce a new one (MS Academic) with hundreds of thousands of *query/relevant document* pairs from the academic domain.

Furthermore, we present a method to estimate the upper bound performance of our RL-based model. Based on the estimated upper bound, we claim that this framework has a strong potential for future improvements.

Here we summarize our main contributions:

- A reinforcement learning framework for automatic query reformulation.
- A simple method to estimate the upper-bound performance of an RL-based model in a given environment.
- A new large dataset with hundreds of thousands of *query/relevant document* pairs.[1]

2 A Reinforcement Learning Approach

2.1 Model Description

In this section we describe the proposed method, illustrated in Fig. 2.

The inputs are a query q_0 consisting of a sequence of words $(w_1, ..., w_n)$ and a candidate term t_i with some context words $(t_{i-k}, ..., t_{i+k})$, where $k \geq 0$ is the context window size. Candidate terms

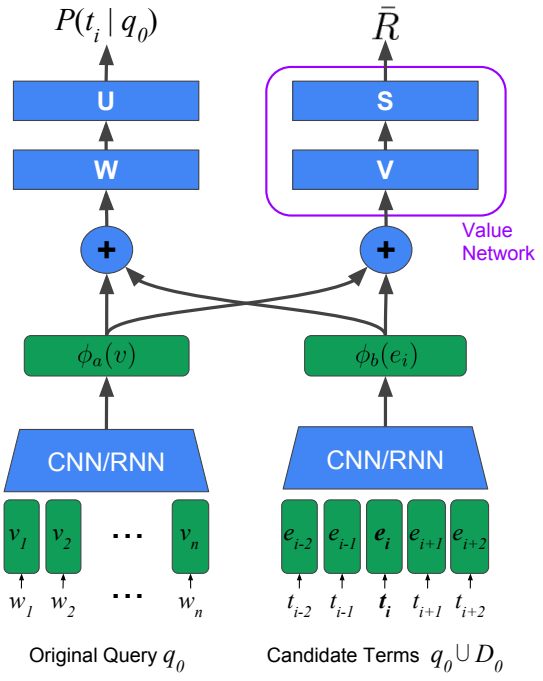

Figure 2: An illustration of our neural network-based reformulator.

are from $q_0 \cup D_0$, the union of the terms in the original query and those from the documents D_0 retrieved using q_0.

We use a dictionary of pretrained word embeddings (Mikolov et al., 2013) to convert the symbolic terms w_j and t_i to their vector representations v_j and $e_i \in \mathbb{R}^d$, respectively. We map out-of-vocabulary terms to an additional vector that is learned during training.

We convert the sequence $\{v_j\}$ to a fixed-size vector $\phi_a(v)$ by using either a Convolutional Neural Network (CNN) followed by a max pooling operation over the entire sequence (Kim, 2014) or by using the last hidden state of a Recurrent Neural Network (RNN).[2]

Similarly, we fed the candidate term vectors e_i to a CNN or RNN to obtain a vector representation $\phi_b(e_i)$ for each term t_i. The convolutional/recurrent layers serve an important role of capturing context information, especially for out-of-vocabulary and rare terms. CNNs can process candidate terms in parallel, and, therefore, are faster for our application than RNNs. RNNs, on the other hand, can encode longer contexts.

Finally, we compute the probability of selecting

[1] The dataset and code to run the experiments are available at https://github.com/nyu-dl/QueryReformulator.

[2] To deal with variable-length inputs in a mini-batch, we pad smaller ones with zeros on both ends so they end up as long as the largest sample in the mini-batch.

t_i as:

$$P(t_i|q_0) = \sigma(U^\mathsf{T} \tanh(W(\phi_a(v)\|\phi_b(e_i)) + b)), \quad (1)$$

where σ is the sigmoid function, $\|$ is the vector concatenation operation, $W \in \mathbb{R}^{d\times 2d}$ and $U \in \mathbb{R}^d$ are weights, and $b \in \mathbb{R}$ is a bias.

At test time, we define the set of terms used in the reformulated query as $T = \{t_i \mid P(t_i|q_0) > \epsilon\}$, where ϵ is a hyper-parameter. At training time, we sample the terms according to their probability distribution, $T = \{t_i \mid \alpha = 1 \wedge \alpha \sim P(t_i|q_0)\}$. We concatenate the terms in T to form a reformulated query q', which will then be used to retrieve a new set of documents D'.

2.2 Sequence Generation

One problem with the method previously described is that terms are selected independently. This may result in a reformulated query that contains duplicated terms since the same term can appear multiple times in the feedback documents. Another problem is that the reformulated query can be very long, resulting in a slow retrieval.

To solve these problems, we extend the model to sequentially generate a reformulated query, as proposed by Buck et al. (2017). We use a Recurrent Neural Network (RNN) that selects one term at a time from the pool of candidate terms and stops when a special token is selected. The advantage of this approach is that the model can remember the terms previously selected through its hidden state. It can, therefore, produce more concise queries.

We define the probability of selecting t_i as the k-th term of a reformulated query as:

$$P(t_i^k|q_0) \propto \exp(\phi_b(e_i)^\mathsf{T} h_k), \quad (2)$$

where h_k is the hidden state vector at the k-th step, computed as:

$$h_k = \tanh(W_a\phi_a(v) + W_b\phi_b(t^{k-1}) + W_h h_{k-1}), \quad (3)$$

where t^{k-1} is the term selected in the previous step and $W_a \in \mathbb{R}^{d\times d}$, $W_b \in \mathbb{R}^{d\times d}$, and $W_h \in \mathbb{R}^{d\times d}$ are weight matrices. In practice, we use an LSTM (Hochreiter and Schmidhuber, 1997) to encode the hidden state as this variant is known to perform better than a vanilla RNN.

We avoid normalizing over a large vocabulary by using only terms from the retrieved documents. This makes inference faster and training practical since learning to select words from the whole vocabulary might be too slow with reinforcement learning, although we leave this experiment for a future work.

2.3 Training

We train the proposed model using REINFORCE (Williams, 1992) algorithm. The per-example stochastic objective is defined as

$$C_a = (R - \bar{R}) \sum_{t\in T} -\log P(t|q_0), \quad (4)$$

where R is the reward and $\bar{R}$ is the baseline, computed by the value network as:

$$\bar{R} = \sigma(S^\mathsf{T} \tanh(V(\phi_a(v)\|\bar{e}) + b)), \quad (5)$$

where $\bar{e} = \frac{1}{N}\sum_{i=1}^{N} \phi_b(e_i)$, $N = |q_0 \cup D_0|$, $V \in \mathbb{R}^{d\times 2d}$ and $S \in \mathbb{R}^d$ are weights and $b \in \mathbb{R}$ is a bias. We train the value network to minimize

$$C_b = \alpha||R - \bar{R}||^2, \quad (6)$$

where α is a small constant (e.g. 0.1) multiplied to the loss in order to stabilize learning. We conjecture that the stability is due to the slowly evolving value network which directly affects the learning of the policy. This effectively prevents the value network to fit extreme cases (unexpectedly high or low reward.)

We minimize C_a and C_b using stochastic gradient descent (SGD) with the gradient computed by backpropagation (Rumelhart et al., 1988). This allows the entire model to be trained end-to-end directly to optimize the retrieval performance.

Entropy Regularization We observed that the probability distribution in Eq.(1) became highly peaked in preliminary experiments. This phenomenon led to the trained model not being able to explore new terms that could lead to a better-reformulated query. We address this issue by regularizing the negative entropy of the probability distribution. We add the following regularization term to the original cost function in Eq. (4):

$$C_H = -\lambda \sum_{t\in q_0\cup D_0} P(t|q_0)\log P(t|q_0), \quad (7)$$

where λ is a regularization coefficient.

3 Related Work

Query reformulation techniques are either based on a global method, which ignores a set of documents returned by the original query, or a local

method, which adjusts a query relative to the documents that initially appear to match the query. In this work, we focus on local methods.

A popular instantiation of a local method is the *relevance model*, which incorporates pseudo-relevance feedback into a language model form (Lavrenko and Croft, 2001). The probability of adding a term to an expanded query is proportional to its probability of being generated by the language models obtained from the original query and the document the term occurs in. This framework has the advantage of not requiring *query/relevant documents* pairs as training data since inference is based on word co-occurrence statistics.

Unlike the relevance model, algorithms can be trained with supervised learning, as proposed by Cao et al. (2008). A training dataset is automatically created by labeling each candidate term as relevant or not based on their individual contribution to the retrieval performance. Then a binary classifier is trained to select expansion terms. In Section 4, we present a neural network-based implementation of this supervised approach.

A generalization of this supervised framework is to *iteratively* reformulate the query by selecting one candidate term at each retrieval step. This can be viewed as navigating a graph where the nodes represent queries and associated retrieved results and edges exist between nodes whose queries are simple reformulations of each other (Diaz, 2016). However, it can be slow to reformulate a query this way as the search engine must be queried for each newly added term. In our method, on the contrary, the search engine is queried with multiple new terms at once.

An alternative technique based on supervised learning is to learn a common latent representation of queries and relevant documents terms by using a *click-through* dataset (Sordoni et al., 2014). Neighboring document terms of a query in the latent space are selected to form an expanded query. Instead of using a *click-through* dataset, which is often proprietary, it is possible to use an alternative dataset consisting of anchor text/title pairs. In contrast, our approach does not require a dataset of paired queries as it learns term selection strategies via reinforcement learning.

Perhaps the closest work to ours is that by Narasimhan et al. (2016), in which a reinforcement learning based approach is used to reformulate queries iteratively. A key difference is that in their work the reformulation component uses domain-specific template queries. Our method, on the other hand, assumes open-domain queries.

4 Experiments

In this section we describe our experimental setup, including baselines against which we compare the proposed method, metrics, reward for RL-based models, datasets and implementation details.

4.1 Baseline Methods

Raw: The original query is given to a search engine without any modification. We evaluate two search engines in their default configuration: Lucene[3] (Raw-Lucene) and Google Search[4] (Raw-Google).

Pseudo Relevance Feedback (PRF-TFIDF): A query is expanded with terms from the documents retrieved by a search engine using the original query. In this work, the top-N TF-IDF terms from each of the top-K retrieved documents are added to the original query, where N and K are selected by a grid search on the validation data.

PRF-Relevance Model (PRF-RM): This is a popular relevance model for query expansion by Lavrenko and Croft (2001). The probability of using a term t in an expanded query is given by:

$$P(t|q_0) = (1-\lambda)P'(t|q_0) + \lambda \sum_{d \in D_0} P(d)P(t|d)P(q_0|d), \quad (8)$$

where $P(d)$ is the probability of retrieving the document d, assumed uniform over the set, $P(t|d)$ and $P(q_0|d)$ are the probabilities assigned by the language model obtained from d to t and q_0, respectively. $P'(t|q_0) = \frac{\text{tf}(t \in q)}{|q|}$, where $\text{tf}(t,d)$ is the term frequency of t in d. We set the interpolation parameter λ to 0.5, following Zhai and Lafferty (2001).

We use a Dirichlet smoothed language model (Zhai and Lafferty, 2001) to compute a language model from a document $d \in D_0$:

$$P(t|d) = \frac{\text{tf}(t,d) + uP(t|C)}{|d| + u}, \quad (9)$$

[3] https://lucene.apache.org/

[4] https://cse.google.com/cse/

where u is a scalar constant ($u = 1500$ in our experiments), and $P(t|C)$ is the probability of t occurring in the entire corpus C.

We use the N terms with the highest $P(t|q_0)$ in an expanded query, where N is a hyper-parameter.

Embeddings Similarity: Inspired by the methods proposed by Roy et al. (2016) and Kuzi et al. (2016), the top-N terms are selected based on the cosine similarity of their embeddings against the original query embedding. Candidate terms come from documents retrieved using the original query (PRF-Emb), or from a fixed vocabulary (Vocab-Emb). We use pretrained embeddings from Mikolov et al. (2013), and it contains 374,000 words.

4.2 Proposed Methods

Supervised Learning (SL): Here we detail a deep learning-based variant of the method proposed by Cao et al. (2008). It assumes that query terms contribute independently to the retrieval performance. We thus train a binary classifier to select a term if the retrieval performance increases beyond a preset threshold when that term is added to the original query. More specifically, we mark a term as relevant if $(R' - R)/R > 0.005$, where R and R' are the retrieval performances of the original query and the query expanded with the term, respectively.

We experiment with two variants of this method: one in which we use a convolutional network for both original query and candidate terms (SL-CNN), and the other in which we replace the convolutional network with a single hidden layer feed-forward neural network (SL-FF). In this variant, we average the output vectors of the neural network to obtain a fixed size representation of q_0.

Reinforcement Learning (RL): We use multiple variants of the proposed RL method. RL-CNN and RL-RNN are the models described in Section 2.1, in which the former uses CNNs to encode query and term features and the latter uses RNNs (more specifically, bidirectional LSTMs). RL-FF is the model in which term and query vectors are encoded by single hidden layer feed-forward neural networks. In the RL-RNN-SEQ model, we add the sequential generator described in Section 2.2 to the RL-RNN variant.

4.3 Datasets

We summarize in Table 1 the datasets.

TREC - Complex Answer Retrieval (TREC-CAR) This is a publicly available dataset automatically created from Wikipedia whose goal is to encourage the development of methods that respond to more complex queries with longer answers (Dietz and Ben, 2017). A query is the concatenation of an article title and one of its section titles. The ground-truth documents are the paragraphs within that section. For example, a query is "*Sea Turtle, Diet*" and the ground truth documents are the paragraphs in the section "*Diet*" of the "*Sea Turtle*" article. The corpus consists of all the English Wikipedia paragraphs, except the abstracts. The released dataset has five predefined folds, and we use the first three as the training set and the remaining two as validation and test sets, respectively.

Jeopardy This is a publicly available Q&A dataset introduced by Nogueira and Cho (2016). A query is a question from the *Jeopardy!* TV Show and the corresponding document is a Wikipedia article whose title is the answer. For example, a query is *"For the last eight years of his life, Galileo was under house arrest for espousing this mans theory"* and the answer is the Wikipedia article titled *"Nicolaus Copernicus"*. The corpus consists of all the articles in the English Wikipedia.

Microsoft Academic (MSA) This dataset consists of academic papers crawled from Microsoft Academic API.[5] The crawler started at the paper Silver et al. (2016) and traversed the graph of references until 500,000 papers were crawled. We then removed papers that had no reference within or whose abstract had less than 100 characters. We ended up with 480,000 papers.

A query is the title of a paper, and the ground-truth answer consists of the papers cited within. Each document in the corpus consists of its title and abstract.[6]

4.4 Metrics and Reward

Three metrics are used to evaluate performance:

Recall@K: Recall of the top-K retrieved documents:

$$\text{R@}K = \frac{|D_K \cap D^*|}{|D^*|}, \tag{10}$$

[5] https://www.microsoft.com/cognitive-services/en-us/academic-knowledge-api

[6] This was done to avoid a large computational overhead for indexing full papers.

Dataset	Corpus	Docs	Queries			Relevant Docs/Query		Words/Doc	
			Train	Valid	Test	Avg.	Std.	Avg.	Std.
TREC-CAR	Wikipedia Paragraphs	3.5M	585k	195k	195k	3.6	5.7	84	68
Jeopardy	Wikipedia Articles	5.9M	118K	10k	10k	1.0	0.0	462	990
MSA	Academic Papers	480k	270k	20k	20k	17.9	21.5	165	158

Table 1: Summary of the datasets.

Method	TREC-CAR			Jeopardy			MSA		
	R@40	P@10	MAP@40	R@40	P@10	MAP@40	R@40	P@10	MAP@40
Raw-Lucene	43.6	7.24	19.6	23.4	1.47	7.40	12.9	7.24	3.36
Raw-Google	-	-	-	30.1	1.92	7.71	-	-	-
PRF-TFIDF	44.3	7.31	19.9	29.9	1.91	7.65	13.2	7.27	3.50
PRF-RM	45.1	7.35	19.5	30.5	1.96	7.64	12.3	7.22	3.38
PRF-Emb	44.5	7.32	19.0	30.1	1.92	7.74	12.2	7.22	3.20
Vocab-Emb	44.2	7.30	19.1	29.4	1.87	7.80	12.0	7.21	3.21
SL-FF	44.1	7.29	19.7	30.8	1.95	7.70	13.2	7.28	3.88
SL-CNN	45.3	7.35	19.8	31.1	1.98	7.79	14.0	7.42	3.99
SL-Oracle	50.8	8.25	21.0	38.8	2.50	9.92	17.3	10.12	4.89
RL-FF	44.1	7.29	20.0	31.0	1.98	7.81	13.9	7.33	3.81
RL-CNN	47.3	7.45	20.3	33.4	**2.14**	8.02	14.9	7.63	4.30
RL-RNN	**47.9**	**7.52**	**20.6**	**33.7**	2.12	**8.07**	**15.1**	**7.68**	**4.35**
RL-RNN-SEQ	47.4	7.48	20.3	33.4	2.13	8.01	14.8	7.63	4.27
RL-Oracle	55.9	9.06	23.0	42.4	2.74	10.3	24.6	12.83	6.33

Table 2: Results on Test sets. We use R@40 as a reward to the RL-based models.

where D_K are the top-K retrieved documents and D^* are the relevant documents. Since one of the goals of query reformulation is to increase the proportion of relevant documents returned, recall is our main metric.

Precision@K: Precision of the top-K retrieved documents:

$$\text{P@}K = \frac{|D_K \cap D^*|}{|D_K|} \tag{11}$$

Precision captures the proportion of relevant documents among the returned ones. Despite not being the main goal of a reformulation method, improvements in precision are also expected with a good query reformulation method. Therefore, we include this metric.

Mean Average Precision: The average precision of the top-K retrieved documents is defined as:

$$\text{AP@}K = \frac{\sum_{k=1}^{K} \text{P@}k \times \text{rel}(k)}{|D^*|}, \tag{12}$$

where

$$\text{rel}(k) = \begin{cases} 1, & \text{if the k-th document is relevant;} \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

The mean average precision of a set of queries Q is then:

$$\text{MAP@}K = \frac{1}{|Q|} \sum_{q \in Q} \text{AP@}K_q, \tag{14}$$

where $\text{AP@}K_q$ is the average precision at K for a query q. This metric values the position of a relevant document in a returned list and is, therefore, complementary to precision and recall.

Reward We use R@K as a reward when training the proposed RL-based models as this metric has shown to be effective in improving the other metrics as well.

SL-Oracle In addition to the baseline methods and proposed reinforcement learning approach, we report two oracle performance bounds. The first oracle is a supervised learning oracle (SL-Oracle). It is a classifier that perfectly selects terms that will increase performance according to the procedure described in Section 4.2. This measure serves as an upper-bound for the supervised methods. Notice that this heuristic assumes that each term contributes independently from all the other terms to the retrieval performance. There may be, however, other ways to explore the dependency of terms that would lead to a higher performance.

	TREC-CAR	Jeopardy	MSA
SL-Oracle	13%	5%	11%
RL-Oracle	29%	27%	31%

Table 3: Percentage of relevant terms over all the candidate terms according to SL- and RL-Oracle.

RL-Oracle Second, we introduce a reinforcement learning oracle (RL-Oracle) which estimates a conservative upper-bound performance for the RL models. Unlike the SL-Oracle, it does not assume that each term contributes independently to the retrieval performance. It works as follows: first, the *validation* or *test* set is divided into N small subsets $\{A_i\}_{i=1}^{N}$ (each with 100 examples, for instance). An RL model is trained on each subset A_i until it overfits, that is, until the reward R_i^* stops increasing or an early stop mechanism ends training.[7] Finally, we compute the oracle performance R^* as the average reward over all the subsets: $R^* = \frac{1}{N}\sum_{i=1}^{N} R_i^*$.

This upper bound by the RL-Oracle is, however, conservative since there might exist better reformulation strategies that the RL model was not able to discover.

4.5 Implementation Details

Search engine We use Lucene and BM25 as the search engine and the ranking function, respectively, for all PRF, SL and RL methods. For Raw-Google, we restrict the search to the *wikipedia.org* domain when evaluating its performance on the Jeopardy dataset. We could not apply the same restriction to the two other datasets as Google does not index Wikipedia paragraphs, and as it is not trivial to match papers from MS Academic to the ones returned by Google Search.

Candidate terms We use Wikipedia articles as a source for candidate terms since it is a well curated, clean corpus, with diverse topics.

At training and test times of SL methods, and at test time of RL methods, the candidate terms are from the first M words of the top-K Wikipedia articles retrieved. We select M and K using grid search on the validation set over $\{50, 100, 200, 300\}$ and $\{1, 3, 5, 7\}$, respectively. The best values are $M = 300$ and $K = 7$. These correspond to the maximum number of terms we could fit in a single GPU.

[7] The subset should be small enough, or the model should be large enough so it can overfit.

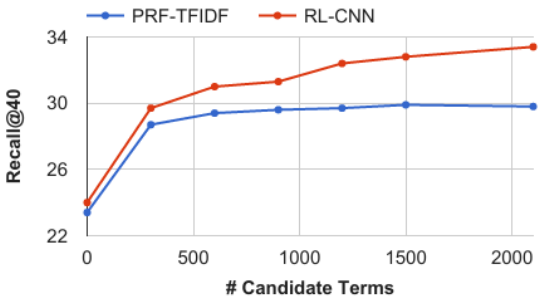

Figure 3: Our RL-based model continues to improve recall as more candidate terms are added, whereas a classical PRF method saturates.

At training time of an RL model, we use only *one* document uniformly sampled from the top-K retrieved ones as a source for candidate terms, as this leads to a faster learning.

For the PRF methods, the top-M terms according to a relevance metric (i.e., TF-IDF for PRF-TFIDF, cosine similarity for PRF-Emb, and conditional probability for PRF-RM) from each of the top-K retrieved documents are added to the original query. We select M and K using grid search over $\{10, 50, 100, 200, 300, 500\}$ and $\{1, 3, 5, 9, 11\}$, respectively. The best values are $M = 300$ and $K = 9$.

Multiple Reformulation Rounds Although our framework supports multiple rounds of search and reformulation, we did not find any significant improvement in reformulating a query more than once. Therefore, the numbers reported in the results section were all obtained from models running two rounds of search and reformulation.

Neural Network Setup For SL-CNN and RL-CNN variants, we use a 2-layer convolutional network for the original query. Each layer has a window size of 3 and 256 filters. We use a 2-layer convolutional network for candidate terms with window sizes of 9 and 3, respectively, and 256 filters in each layer. We set the dimension d of the weight matrices W, S, U, and V to 256. For the optimizer, we use ADAM (Kingma and Ba, 2014) with $\alpha = 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. We set the entropy regularization coefficient λ to 10^{-3}.

For RL-RNN and RL-RNN-SEQ, we use a 2-layer bidirectional LSTM with 256 hidden units in each layer. We clip the gradients to unit norm. For RL-RNN-SEQ, we set the maximum possible

number of generated terms to 50 and we use beam search of size four at test time.

We fix the dictionary of pre-trained word embeddings during training, except the vector for out-of-vocabulary words. We found that this led to faster convergence and observed no difference in the overall performance when compared to learning embeddings during training.

5 Results and Discussion

Table 2 shows the main result. As expected, reformulation based methods work better than using the original query alone. Supervised methods (SL-FF and SL-CNN) have in general a better performance than unsupervised ones (PRF-TFIDF, PRF-RM, PRF-Emb, and Emb-Vocab), but perform worse than RL-based models (RL-FF, RL-CNN, RL-RNN, and RL-RNN-SEQ).

RL-RNN-SEQ performs slightly worse than RL-RNN but produces queries that are three times shorter, on average (15 vs 47 words). Thus, RL-RNN-SEQ is faster in retrieving documents and therefore might be a better candidate for a production implementation.

The performance gap between the oracle and best performing method (Table 2, RL-Oracle vs. RL-RNN) suggests that there is a large room for improvement. The cause for this gap is unknown but we suspect, for instance, an inherent difficulty in learning a good selection strategy and the partial observability from using a black box search engine.

5.1 Relevant Terms per Document

The proportion of relevant terms selected by the SL- and RL-Oracles over the total number of candidate terms (Table 3) indicates that only a small subset of terms are useful for the reformulation. Thus, we may conclude that the proposed method was able to learn an efficient term selection strategy in an environment where relevant terms are infrequent.

5.2 Scalability: Number of Terms vs Recall

Fig. 3 shows the improvement in recall as more candidate terms are provided to a reformulation method. The RL-based model benefits from more candidate terms, whereas the classical PRF method quickly saturates. In our experiments, the best performing RL-based model uses the maximum number of candidate terms that we could fit on a single GPU. We, therefore, expect further improvements with more computational resources.

Query	Top-3 Retrieved Documents
(Original) *The Cross Entropy Method for Fast Policy Search*	*-The Cross Entropy Method for Network Reliability Estim.* ***-Robot Weightlifting by Direct Policy Search*** *-Off-policy Policy Search*
(Reformulated) *Cross Entropy Fast Policy Reinforcement Learning policies global search optimization biased*	***-Near Optimal Reinforcement Learning in Polynom. Time*** *-The Cross Entropy Method for Network Reliability Estim.* ***-Robot Weightlifting by Direct Policy Search***
(Original) *Daikon Cultivation*	"*...many types of pickles are made with daikon, includ...*" "***Certain varieties of daikon can be grown as a winter...***" "*In Chinese cuisine, turnip cake and chai tow kway...*"
(Reformulated) *Daikon Cultivation root seed grow fast-growing Chinese leaves*	"*...many types of pickles are* "*made with daikon, includ...*" "***Certain varieties of daikon can be grown as a winter...***" "***The Chinese and Indian varieties tolerate higher....***"

Table 4: Top-3 retrieved documents using the original query and a query reformulated by our RL-CNN model. In the first example, we only show the titles of the retrieved MSA papers. In the second example, we only show some words of the retrieved TREC-CAR paragraphs. **Bold** corresponds to ground-truth documents.

5.3 Qualitative Analysis

We show two examples of queries and the probabilities of each candidate term of being selected by the RL-CNN model in Fig. 4.

Notice that terms that are more related to the query have higher probabilities, although common words such as "*the*" are also selected. This is a consequence of our choice of a reward that does

Trained on	Selected Terms
TREC-CAR	*serves american national Winsted accreditation*
Jeopardy	*Tunxis Quinebaug Winsted NCCC*
MSA	*hospital library arts center cancer center summer programs*

Table 5: Given the query *"Northwestern Connecticut Community College"*, models trained on different tasks choose different terms.

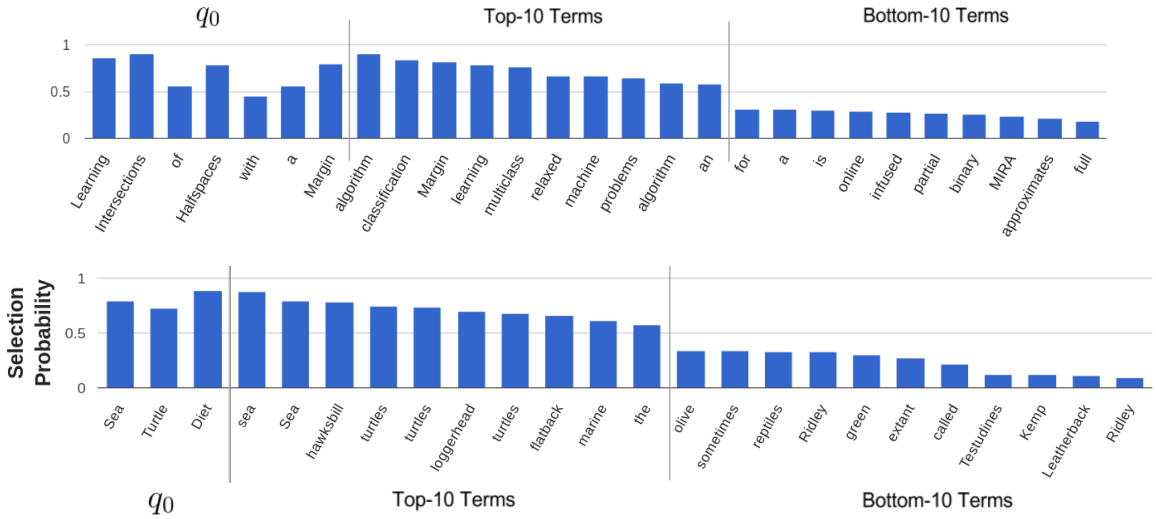

Figure 4: Probabilities assigned by the RL-CNN to candidate terms of two sample queries: "*Learning Intersections of Halfspaces with a Margin*" (top) and "*Sea Turtle Diet*" (bottom). We show the original query terms and the top-10 and bottom-10 document terms with respect to their probabilities.

not penalize the selection of neutral terms.

In Table 4 we show an original and reformulated query examples extracted from the MS Academic and TREC-CAR datasets, and their top-3 retrieved documents. Notice that the reformulated query retrieves more relevant documents than the original one. As we conjectured earlier, we see that a search engine tends to return a document simply with the largest overlap in the text, necessitating the reformulation of a query to retrieve semantically relevant documents.

Same query, different tasks We compare in Table 5 the reformulation of a sample query made by models trained on different datasets. The model trained on TREC-CAR selects terms that are similar to the ones in the original query, such as "*serves*" and "*accreditation*". These selections are expected for this task since similar terms can be effective in retrieving similar paragraphs. On the other hand, the model trained on Jeopardy prefers to select proper nouns, such as "*Tunxis*", as these have a higher chance of being an answer to the question. The model trained on MSA selects terms that cover different aspects of the entity being queried, such as "*arts center*" and "*library*", since retrieving a diverse set of documents is necessary for the task the of citation recommendation.

5.4 Training and Inference Times

Our best model, RL-RNN, takes 8-10 days to train on a single K80 GPU. At inference time, it takes approximately one second to reformulate a batch of 64 queries. Approximately 40% of this time is to retrieve documents from the search engine.

6 Conclusion

We introduced a reinforcement learning framework for task-oriented automatic query reformulation. An appealing aspect of this framework is that an agent can be trained to use a search engine for a specific task. The empirical evaluation has confirmed that the proposed approach outperforms strong baselines in the three separate tasks. The analysis based on two oracle approaches has revealed that there is a meaningful room for further development. In the future, more research is necessary in the directions of (1) iterative reformulation under the proposed framework, (2) using information from modalities other than text, and (3) better reinforcement learning algorithms for a partially-observable environment.

Acknowledgements

RN is funded by Coordenao de Aperfeioamento de Pessoal de Nvel Superior (CAPES). KC thanks support by Facebook, Google and NVIDIA. This work was partly funded by the Defense Advanced Research Projects Agency (DARPA) D3M program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

Christian Buck, Jannis Bulian, Massimiliano Ciaramita, Andrea Gesmundo, Neil Houlsby, Wojciech Gajewski, and Wei Wang. 2017. Ask the right questions: Active question reformulation with reinforcement learning. *arXiv preprint arXiv:1705.07830*.

Guihong Cao, Jian-Yun Nie, Jianfeng Gao, and Stephen Robertson. 2008. Selecting good expansion terms for pseudo-relevance feedback. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 243–250. ACM.

Fernando Diaz. 2016. Pseudo-query reformulation. In *European Conference on Information Retrieval*, pages 521–532. Springer.

Laura Dietz and Gamari Ben. 2017. Trec car: A data set for complex answer retrieval. *http://trec-car.cs.unh.edu*.

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

Jeff Huang and Efthimis N Efthimiadis. 2009. Analyzing and evaluating query reformulation strategies in web search logs. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 77–86. ACM.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Saar Kuzi, Anna Shtok, and Oren Kurland. 2016. Query expansion using word embeddings. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1929–1932. ACM.

Victor Lavrenko and W Bruce Croft. 2001. Relevance based language models. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 120–127. ACM.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Karthik Narasimhan, Adam Yala, and Regina Barzilay. 2016. Improving information extraction by acquiring external evidence with reinforcement learning. *arXiv preprint arXiv:1603.07954*.

Rodrigo Nogueira and Kyunghyun Cho. 2016. End-to-end goal-driven web navigation. In *Advances in Neural Information Processing Systems*, pages 1903–1911.

Dwaipayan Roy, Debjyoti Paul, Mandar Mitra, and Utpal Garain. 2016. Using word embeddings for automatic query expansion. *arXiv preprint arXiv:1606.07608*.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1988. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Alessandro Sordoni, Yoshua Bengio, and Jian-Yun Nie. 2014. Learning concept embeddings for query expansion by quantum entropy minimization. In *AAAI*, pages 1586–1592.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Jinxi Xu and W Bruce Croft. 1996. Query expansion using local and global document analysis. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 4–11. ACM.

Chengxiang Zhai and John Lafferty. 2001. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 334–342. ACM.