From Neural Re-Ranking to Neural Ranking: Learning a Sparse Representation for Inverted Indexing

Hamed Zamani
University of Massachusetts Amherst
zamani@cs.umass.edu

Mostafa Dehghani
University of Amsterdam
dehghani@uva.nl

W. Bruce Croft
University of Massachusetts Amherst
croft@cs.umass.edu

Erik Learned-Miller
University of Massachusetts Amherst
elm@cs.umass.edu

Jaap Kamps
University of Amsterdam
kamps@uva.nl

ABSTRACT

The availability of massive data and computing power allowing for effective data driven neural approaches is having a major impact on machine learning and information retrieval research, but these models have a basic problem with efficiency. Current neural ranking models are implemented as multistage rankers: for efficiency reasons, the neural model only *re-ranks* the top ranked documents retrieved by a first-stage efficient ranker in response to a given query. Neural ranking models learn dense representations causing essentially every query term to match every document term, making it highly inefficient or intractable to rank the whole collection. The reliance on a first stage ranker creates a dual problem: First, the interaction and combination effects are not well understood. Second, the first stage ranker serves as a "gate-keeper" or filter, effectively blocking the potential of neural models to uncover new relevant documents.

In this work, we propose a *standalone* neural ranking model (SNRM) by introducing a *sparsity property* to learn a latent sparse representation for each query and document. This representation captures the semantic relationship between the query and documents, but is also sparse enough to enable constructing an inverted index for the whole collection. We parameterize the sparsity of the model to yield a retrieval model as efficient as conventional term based models. Our model gains in efficiency without loss of effectiveness: it not only outperforms the existing term matching baselines, but also performs similarly to the recent re-ranking based neural models with dense representations. Our model can also take advantage of pseudo-relevance feedback for further improvements. More generally, our results demonstrate the importance of sparsity in neural IR models and show that dense representations can be pruned effectively, giving new insights about essential semantic features and their distributions.

ACM Reference Format:
Hamed Zamani, Mostafa Dehghani, W. Bruce Croft, Erik Learned-Miller, and Jaap Kamps. 2018. From Neural Re-Ranking to Neural Ranking: Learning a Sparse Representation for Inverted Indexing. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271800

1 INTRODUCTION

Retrieving unstructured documents in response to a natural language query is the core task in information retrieval (IR). Due to the importance of this task, the IR community has put a significant emphasis on designing efficient and effective retrieval models since the early years. The recent and successful development of deep neural networks for various tasks has also impacted IR applications. In particular, neural ranking models (NRMs) have recently shown significant improvements in a wide range of IR applications, such as ad-hoc retrieval [16, 20, 34, 49], question answering [51], context-aware retrieval [55], mobile search [1], and product search [46]. Most of the existing neural ranking models have a specific property in common: they are employed for *re-ranking* a small set of potentially relevant documents for a given query, provided by an efficient first stage ranker. In other words, since most neural ranking models rely on semantic matching that can be achieved using distributed dense representations, computing the retrieval score for all the documents in a large-scale collection is generally infeasible.

A multistage ranker is a pragmatic approach that seems to combine the advantages of efficiency and effectiveness from the neural model. However, the multistage setup may be inefficient due to the multiple, stacked rankers working at query time, and may reduce effectiveness by limiting the set of documents continuing to the next stage and by introducing and propagating errors. The representations of documents and queries are what makes the traditional term based ranking models fast, and the neural ranking models slow. Queries are short and terms have a highly skewed Zipfian distribution making each term relatively selective, resulting in a simple join over very few relatively short posting lists [48]. In contrast, dense representations have an almost uniform distribution, with every term (to some degree) matching essentially all documents—similar to extreme stopwords that we cannot filter out.

Our approach addresses this head-on: by enforcing and rewarding sparsity in the representation learning, we create a latent representation that aims to capture meaningful semantic relations while still parsimoniously matching documents. This is illustrated in Figure 1, showing that the Zipfian distribution of the term space is matching far fewer documents than the dense representation returning collection-length posting lists for every term, dramatically increasing index size and query processing time. However, the latent sparse representation proposed in this paper mimics the posting list length distribution of the term based model, even matching fewer documents than term based models. That is, unlike existing neural ranking models, we propose to learn *high-dimensional sparse* representations for query and documents in order to allow for an inverted

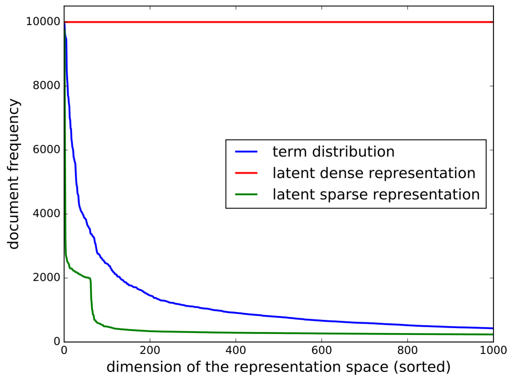

Figure 1: The document frequency for the top 1000 dimensions in the actual term space (blue), the latent dense space (red), and the latent sparse space (green) for a random sample of 10k documents from the Robust collection.

index based *standalone neural ranker.* Our model does not require a first stage ranker and can retrieve documents from a large-scale collection as efficient as conventional term matching models, such as TF-IDF, BM25, or query likelihood models.

Our main goal is learning representations for documents and queries that result in better matching compared to the original term vectors and exact matching models, while we still inherit the efficiency rooted in the sparsity of those representations. So there are two objectives, introducing sparsity and capturing latent semantic meanings. Based on these two objectives, given a document or a query, the proposed model first maps each ngram to a low-dimensional dense representation to compress the information and learn the low dimensional manifold of the data, and then learns a function to transform it to a high-dimensional representation pursuing the sparsity as a desired characteristic for these representations. By aggregating the sparse ngram representations, we obtain a sparse representation for a text with an arbitrary length, whose sparsity is a function of the input length; this implies higher sparsity for queries in comparison with documents, which is desired to have an efficient retrieval model. We achieve a sparsity ratio in the learned representations that is comparable to the sparsity ratio in original term vectors of documents and queries, while we are also able to better capture the semantic relevance of queries and documents using simple and efficient vector space matching functions on the learned representations.

Once our latent sparse representation is trained, we initiate an inverted index construction phase that looks at each dimension of the learned representation as a "latent term" and builds an inverted index from each latent term to each document of the collection. This is an offline process and the constructed inverted index allows for efficient retrieval. At query time, we transform a given query to the learned latent high-dimensional space, and obtain its sparse representation. Given the small number of non-zero elements of the query representation and the constructed inverted index, we are able to retrieve documents from a large collection efficiently. We also study pseudo-relevance feedback in the learned semantic space. We train our models using weak supervision, an *unsupervised* learning approach that relies on an existing retrieval model, such as query likelihood, to generate a large volume of training data with weak labels. Weak supervision has been recently proven to be effective for learning neural ranking models [16, 17] and learning relevance-based word embedding vectors [52]. We conduct extensive experiments with SNRM on newswire and large-scale web collections and demonstrate the effectiveness of the proposed model. In summary, we show that SNRM not only performs as efficiently as term matching models, e.g., query likelihood, but also performs as effectively as re-ranking based neural ranking models. Our model can take advantage of pseudo-relevance feedback and significantly outperform competitive baselines.

2 RELATED WORK

In this section, we discuss related work on neural ranking models, weak supervision, attempts on ranking instead of re-ranking, and sparse coding for representation learning.

Neural Ranking Models Several recent studies have applied deep neural network methods to a number of IR tasks, including question answering [51], ad-hoc retrieval [34, 49], and context-aware ranking [55]. Neural ranking models can be partitioned into early and late combination models [16]. They can also be categorized based on whether they focus on lexical matching or learning text representations for semantic matching [20, 34].

The early combination models are designed based on the interactions between query and document as the networks' input. For instance, DRMM [20] models the interaction between query and document contents based on histogram analysis. As another example, DeepMatch [27] computes the matching score for a text pair by considering word sequences. The local component of the duet model in [34] and the neural ranking models proposed in [16, 49] are the other examples of early combination models.

The late combination models, on the other hand, first learn query and document representations and then compute the relevance score by applying a matching function on the learned representations. DSSM [21] is example of a late combination model that uses a fully-connected feed-forward network for representation learning. C-DSSM [43] is an extension of this model that makes use of convolutional neural networks to capture term dependencies. The distributed component of the duet model [34] also uses a similar architecture for learning document representation. Most recently, NRM-F [58] takes advantage of convolutional networks to represent semi-structured documents. We refer the reader to [33] that provides an overview of neural ranking models.

Our model also belongs to the second category. The main difference between our model and the existing ones is the sparseness of the learned representations that allows us to construct an inverted index for efficiency purposes.

Weak Supervision Limited training data has been a perennial problem in information retrieval, and many machine learning-related domains [57]. This has motivated researchers to explore building models using *pseudo-labels.* For example, pseudo-relevance feedback (PRF) [4] assumes that the top retrieved documents in response to a given query are relevant to the query. Although this assumption does not necessarily hold, PRF has been proven to be effective in many retrieval settings [24, 41, 59]. Building pseudo-collections and simulated queries for various IR tasks could be considered as another set of approaches that tackle this issue [3, 5].

As widely known, deep neural networks often require a large volume of training data. Recently, training neural IR models based on pseudo-labels has been shown to produce successful results [16, 52].

This learning approach is called *weak supervision.* Dehghani et al. [16] proposed training a neural ranking model for the ad-hoc retrieval task based on the labels generated by an existing retrieval model, such as BM25. Zamani and Croft [52] argued that the objective functions of the general-purpose word embedding models, such as word2vec [32], are not necessarily equivalent to the objective that we seek in information retrieval. They proposed training of relevance-based word embeddings based on the relevance models [24] as the weak label. Following these studies, the idea of training neural IR models with weak supervision has been further employed [14, 15, 26, 28, 37, 56]. Most recently, Zamani and Croft [53] provided a theoretical foundation for explaining the successful empirical results achieved by weakly supervised IR models. Moreover, the weak supervision idea has been also used for improving efficiency in IR models [8]. In this paper, we train our model with weak supervision by considering the query likelihood model [40] as the weak labeler.

Ranking Instead of Re-Ranking As mentioned above, most neural ranking models learn a dense latent representation for query and documents [16, 34, 58] to be able to do semantic matching. They re-rank a small set of documents retrieved by a first stage ranker. Since it is not possible to score all documents in each collection for every query, one idea to move from the re-ranking scenario to ranking is to operate in the vector space and rely on k-nearest neighbors search to retrieve the top k documents [47]. However, the exact brute-force k-NN is practically infeasible and an approximation of k-NN [35] is used to be able to make use of dense representation in the ranking scenario [6]. Recently, a neural ranking model based on approximate k-NN has been proposed in [47]. However, the proposed model is not scalable to large-scale collections. In this paper, as an alternative to approximate k-NN models, we focus on learning sparse representations for inverted indexing which is the industry standard because of their known efficiency [10].

Sparse Coding Sparsity is a desirable characteristic for representing data as not only it is an efficient model of data representation, but also captures the generation process of most real-world data, in particular textual data [7, 30]. In the context of deep neural networks, sparsity is often achieved via regularization [44] and activation functions [2]. In this paper, we also minimize the $L1$-norm as a regularization term to introduce sparsity into the learned representations.

3 STANDALONE NEURAL RANKING MODEL

In this section, we introduce SNRM as a standalone neural ranking model based on learning a latent sparse representation. Section 3.1 describes the desirable properties of the model. Section 3.2 details the neural network architecture. The general design of our model consists of three phases: i) the training phase (Section 3.3), ii) the offline inverted index construction phase (Section 3.4), and iii) the retrieval phase with or without feedback (Sections 3.5 and 3.6).

3.1 Design Desiderata

Designing a standalone neural ranking model that can retrieve documents from a large-scale collection, instead of re-ranking a small set of documents returned by a first stage ranker, could potentially be used in various retrieval engines. Conventional term matching retrieval models, such as TF-IDF, BM25, or query likelihood, derive query and document representations based on the atomic units of natural languages (e.g., words); hence they carry the desirable property of *sparsity* which helps efficient retrieval on large-scale collections. They each use an inverted index built on the search collection. Based on the idea of using an inverted index for efficient retrieval, this paper introduces SNRM, a neural network model for learning standalone rankers based on *latent sparse* representations.

Similar to representation-focused neural ranking models, e.g., [16, 21, 43, 58], our neural framework consists of three major components: the query representation ϕ_Q, the document representation ϕ_D, and the matching function ψ. The retrieval score for each query-document pair (q,d) is then computed as follows:

$$\text{retrieval score}(q,d) = \psi\ (\phi_Q(q), \phi_D(d)) \qquad (1)$$

Our goal to build a responsive standalone ranker leads to a number of requirements on these three components:

- ϕ_D is a query independent component, and thus can be computed offline. In contrast to previous neural ranking models that learn low-dimensional dense representations, ϕ_D should output a high-dimensional sparse vector. This will allow us to construct an inverted index based on the learned representations.
- The components ϕ_Q and ψ should be run at query or inference time (i.e., when producing the ranked output). To obtain a real-time ranking model, it is necessary to minimize the amount of computation in these two components.
- The i^{th} element of ϕ_Q and ϕ_D should represent the same latent feature. In other words, the two components ϕ_Q and ϕ_D should provide representations in the same semantic space, which also implies $|\phi_Q| = |\phi_D|$. Considering the first property, this property also means that the query representation should be sparse.
- The matching function ψ should return zero if there is no overlap between the indices of non-zero elements in ϕ_Q and ϕ_D. This property, in addition to the previous one, is necessary to build and use an inverted index for efficient retrieval.
- Functions ϕ_Q and ϕ_D use the full capacity of the output space and do not map queries and documents to a particular subspace. In other words, for each dimension in the output space, there exists at least one query q or document d such that $\phi_Q(q)$ or $\phi_D(d)$ leads to a non-zero value for that dimension.

We claim that if the design of a neural ranking model satisfies the above properties, and the learned query and document representations are sufficiently sparse, we are then able to construct an inverted index from the learned representation by ϕ_D and retrieve documents in response to a given query as a standalone ranking, without the need for a first stage retrieval model.

3.2 Network Architecture

Since the representations learned by ϕ_Q and ϕ_D in SNRM should represent the same semantic space, we design our architecture to share parameters between these components. Furthermore, we need different levels of sparsity in the query and document representations. This is mainly motivated by efficiency, as a key feature of a standalone ranker, since the matching function will compute the retrieval score for all the documents that have a non-zero value for at least one of the non-zero latent elements in the query representation. So the sparser the representation of the query is, the less computation is expected. Furthermore, queries are intuitively much shorter than documents and contain less information, so to express queries, it makes sense to have fewer non-zero elements in their representations compared to the document representations. Using a simple fully-connected

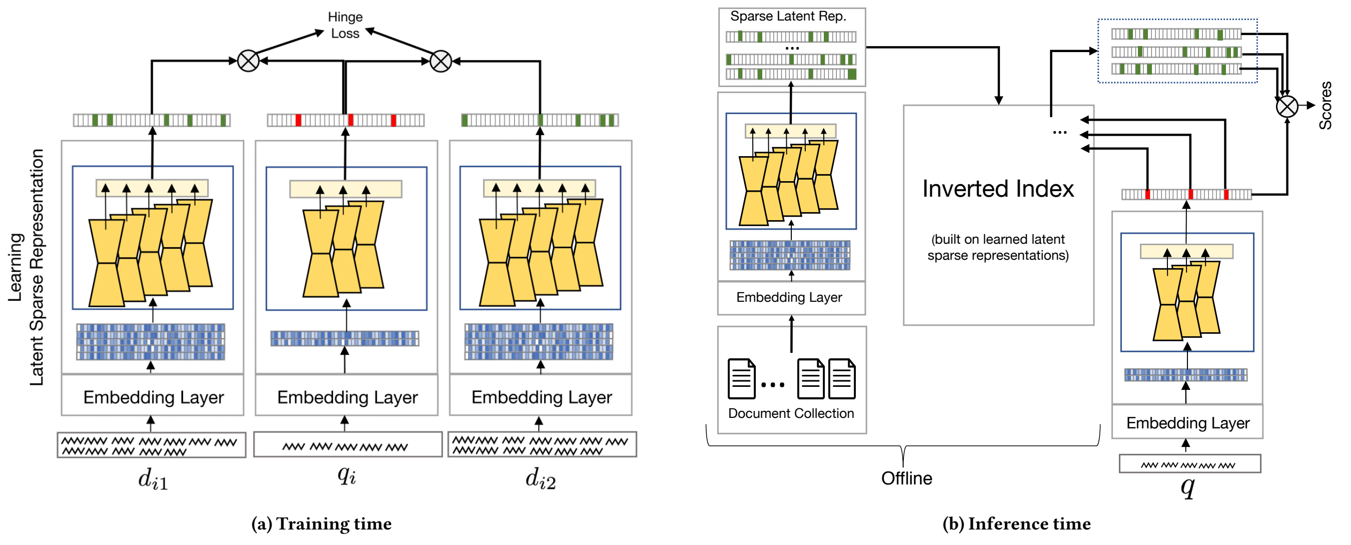

Figure 2: General schema of the proposed SNRM model.

feed-forward network for implementing these components is not an appropriate solution as in this case (due to the shared parameters between ϕ_Q and ϕ_D) both query and document representations would become similar in terms of sparsity ratio, i.e., percentage of zero elements (see Equation (4)). This results in long queries in the learned latent space, leading to an expensive matching function.

Based on this argument, we need a representation learning model in which the representation sparsity is a function of the input length. We propose an architecture based on ngram representation learning. The intuition behind our model is that we first learn a sparse representation for each continuous n words in the given document or query. The learned sparse representations are then aggregated via average pooling. In fact, our document representation (and similarly our query representation) is obtained as follows:

$$\phi_D(d) = \frac{1}{|d|-n+1} \sum_{i=1}^{|d|-n+1} \phi_{\text{ngram}}(w_i, w_{i+1}, \cdots, w_{i+n-1}) \qquad (2)$$

where $w_1, w_2, \cdots, w_{|d|}$ denote the terms appearing in document d with the same order. ϕ_{ngram} learns a sparse representation for the given ngram. The query representation is also computed using the same function. This approach provides two important advantages:

- The number of representations averaged in Equation (2) has a direct relationship with the input document/query length. Therefore, the level of the sparsity in the final learned representations depends on the length of the input text. This results in more sparse representations for queries in comparison to documents, which is desired.
- Using the sliding window for encoding the input words as a set of ngrams helps to capture the local interaction among terms, hence the model considers term dependencies. Term dependencies have been widely known to be useful for improving the retrieval performance [31].

We model our ngram representation function ϕ_{ngram} by a fully-connected feed-forward network that reads the input words using a sliding window over the sequence of their embeddings and encodes their ngrams. To this end, we first collect the embedding vectors for each term in the given ngram from an embedding matrix $\mathcal{E} \in \mathbb{R}^{|V| \times m}$ where V and m denote the vocabulary set and the embedding dimensionality, respectively. As discovered in [16, 49, 52], using relevance-based embedding vectors trained to optimize IR objectives can lead to significant improvements, compared to those trained by general-purpose word embedding vectors, such as word2vec [32] or GloVe [39]. Therefore, $\mathcal{E}$ is part of our network parameters that is learned in an end-to-end training setting. The collected n embedding vectors for the given ngram are then concatenated and fed into a stack of fully-connected layers. These fully-connected layers have an hourglass structure that forces the information of the input data to be compressed and passed through a small number of units in the middle layers that are meant to learn the low dimensional manifold of the data. Then the number of hidden units increases in upper layers to give us a high-dimensional output, e.g., 20,000. Note that in terms of the structure and the dimension of layers, the model looks like an autoencoder. However, unlike autoencoders, we have no reconstruction loss. We will discuss our training in Section 3.3. We employ rectified linear unit (ReLU) as the activation function in our model.

Our neural architecture for query and document representation can also be seen as a multi-layer one-dimensional convolutional network in which the window sizes for all layers except the first one are set to 1. The window size for the first layer is equal to n, and the strides are all set to 1. Figure 3 illustrates how a sequence of words is mapped to a latent sparse representation. This building block of the model is then used in the offline process of encoding documents to sparse latent representations to build an inverted index and also used at inference time to encode submitted queries. The architecture of this block is parallelizable, which supports an efficient procedure for encoding queries at inference time.

We define the matching function ψ as the dot product of the query and document representations. It can be simply proved that dot product has all the properties mentioned earlier in Section 3. Making the model as efficient as possible is our main reason behind using such a simple function to measure the query-document relevance scores.

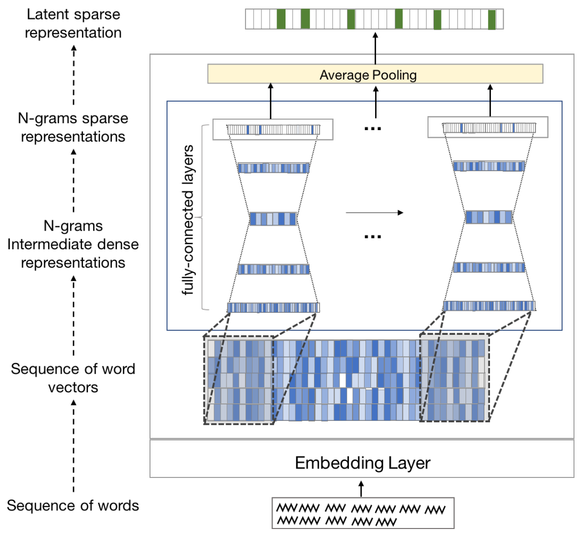
Latent sparse representation
N-grams sparse representations
N-grams Intermediate dense representations
Sequence of word vectors
Sequence of words
Average Pooling
fully-connected layers
Embedding Layer

Figure 3: Learning a latent sparse representation for a document.

This component plays a key role in efficiency of the model, since it is frequently used at inference time (see Section 3.5).

3.3 Training

To train the SNRM framework we have two objectives: the retrieval objective with the goal of improving retrieval accuracy and the sparsity objective with the goal of increasing sparseness in the learned query and document representations.

Let $T = \{(q_1, d_{11}, d_{12}, y_1), \cdots, (q_N, d_{N1}, d_{N2}, y_N)\}$ denote a set of N training instances; each containing a query string q_i, two document candidates d_{i1} and d_{i2}, and a label $y_i \in \{-1, 1\}$ indicating which document is more relevant to the query. In the following, we explain how we optimize our model to achieve both of the mentioned goals.

Retrieval Objective We train our model using a pairwise setting as depicted in Figure 2a. We employ hinge loss (max-margin loss function) that has been widely used in the learning to rank literature for pairwise models [25]. Hinge loss is a linear loss function that penalizes examples violating a margin constraint. The hinge loss for the i^{th} training instance is defined as follows:

$$\mathcal{L} = \max\left\{0, \epsilon - y_i \left[\psi(\phi_Q(q_i), \phi_D(d_{i1})) - \psi(\phi_Q(q_i), \phi_D(d_{i2}))\right]\right\} \quad (3)$$

where ϵ is a hyper-parameter determining the margin of hinge loss.

Sparsity Objective In addition to improving the retrieval accuracy, our model aims at maximizing the *sparsity ratio*, which is defined as follows:

$$\text{sparsity ratio } (\vec{v}) = \frac{\text{total number of zero elements in } \vec{v}}{|\vec{v}|} \quad (4)$$

Defining $0^0 = 0$, maximizing the sparsity ratio is equivalent to minimizing the L_0 norm:

$$L_0(\vec{v}) = \sum_{i=1}^{|\vec{v}|} |\vec{v}_i|^0 \quad (5)$$

Therefore, minimizing the L_0 norm for the final query and document representations is also one of our main objectives. However, the L_0 norm is non-differentiable, which makes it impossible to train our model with the backpropagation algorithm. In fact, L_0 minimization is a non-convex optimization problem, and even finding a solution that approximates the true minimum for L_0 is NP-hard [36]. Therefore, a tractable surrogate loss function should be considered. An alternative would be minimizing L_1 norm (i.e., $L_1(\vec{v}) = \sum_{i=1}^{|\vec{v}|} |\vec{v}_i|$). Although it is clear that we can minimize L_1 as a term in our loss function, it is not immediately obvious how minimizing L_1 would lead to sparsity in the query and document representations. Employing the L_1 norm to promote sparsity has a long history, dating back at least to 1930s for the Fourier transform extrapolation from partial observations [11]. L_1 has been also employed in the information theory literature for recovering band-limited signals [19]. Later on, sparsity minimization has received significant attention as a method for hyper-parameter optimization in regression, known as the Lasso [45].

The theoretical justification for the hypothesis that L_1 minimization would lead to sparsity in our model relies on two points: (1) the choice of rectified linear unit (ReLU) as the activation function forces the non-positive elements to be zero ($\text{ReLU}(x) = \max\{0, x\}$), and (2) the gradient of $L_1(\vec{v})$ for an element of $\vec{v}$ is constant and thus independent of its value. Therefore, the gradient optimization approach used in the backpropagation algorithm [42] reduces the elements of the query and document representation vectors independent of their values. This moves small values toward zero and thus the desired sparsity is obtained.

Loss Function The final loss function for the i^{th} training instance is defined as follows:

$$\mathcal{L}(q_i, d_{i1}, d_{i2}, y_i) + \lambda\, L_1(\phi_Q(q_i) || \phi_D(d_{i1}) || \phi_D(d_{i2})) \quad (6)$$

where $||$ means concatenation. The hyper-parameter λ controls the sparsity of the learned representations.

Training with Weak Supervision We train our model with weak supervision, an unsupervised learning approach that benefits from the pseudo-labels obtained by an existing retrieval model, called weak labeler. Training neural models with weak supervision has been shown to be effective for a set of IR tasks, including ranking [16] and learning relevance-based word embedding [52], as well as a set of NLP tasks, including sentiment classification [18]. In more detail, given a large set of queries and a collection of documents, we first retrieve the documents for each training query q_i using the query likelihood retrieval model [40] with Dirichlet prior smoothing [60] as our weak labeler. Each training instance $(q_i, d_{i1}, d_{i2}, y_i)$ is then obtained by sampling two candidate documents from the result list or one from the result list and one random negative sample from the collection. y_i is defined as:

$$y_i = \text{sign}(p_{QL}(q_i|d_{i1}) - p_{QL}(q_i|d_{i2})) \quad (7)$$

where p_{QL} denotes the query likelihood probability.

3.4 Inverted Index Construction

The training phase in SNRM is followed by an inverted index construction phase. In this phase, as shown in Figure 2b, we first feed each document in the collection into the trained document representation component. We look at each index of the learned representation as a "latent term". In other words, let M denote the dimensionality of document representation, e.g., 20,000. Thus, we assume that there exist M latent terms. Therefore, if the i^{th} element of $\phi_D(d)$ is non-zero, then the document d would be added to the inverted index for

Table 1: Collection statistics.

ID	collection	queries (title only)	#docs	avg doc length	#qrels
Robust	TREC Disks 4 & 5 minus CR	TREC 2004 Robust Track, topics 301-450 & 601-700	528k	254	17,412
ClueWeb	ClueWeb 09 - Category B	TREC 2009-2012 Web Track, topics 1-200	50m	1,506	18,771

the latent term i. The value of this element is the weight of the latent term i in the learned high-dimensional latent vector space.

Since documents of a collection can be assumed to be independent, the inverted index construction phase will be memory efficient, and we can feed documents to the document representation component using mini-batches. Note that in case of incorporating new documents, there is no need to train the network from scratch; we can obtain representation for every new document. In Section 4.4, we have some experiments that support this claim. However, due to the temporal property of natural languages and invention of new vocabulary terms, the model can be re-trained, periodically. This indicates that SNRM is applicable in real-world scenarios.

3.5 Retrieval (Inference)

As illustrated in Figure 2b, at inference time, we first feed the query q into the query representation sub-network ϕ_Q to obtain the learned sparse representation $\vec{q}$. Given the dot product definition of ψ at training time, the retrieval score for each document d at inference time is computed as:

$$\text{retrieval score}(q,d) = \sum_{\vec{q}_i|>0} \vec{q}_i \vec{d}_i \tag{8}$$

which is a summation over the non-zero elements of $\vec{q}$. With the constructed inverted index, the documents with non-zero elements in the i^{th} index can be retrieved efficiently. Similar to the existing term matching retrieval models, such as query likelihood and BM25, retrieval scores can be computed via the Map Reduce framework [12].

3.6 Pseudo-Relevance Feedback

Pseudo-relevance feedback (PRF) is an approach that assumes that the top retrieved documents in response to a given query are relevant, and uses those documents for query expansion. This is also known as a local approach for query expansion [50]. PRF has been proven to be highly effective in various retrieval tasks [10]. We can also take advantage of PRF in our SNRM model. To do so, we use the Rocchio's relevance feedback algorithm [41] for vector space models. Let $\{d_1, d_2, \cdots, d_k\}$ be the top k documents retrieved by SNRM in response to the query q. The updated query vector is computed as:

$$\vec{q}^* = \vec{q} + \alpha \frac{1}{k} \sum_{i=1}^{k} \vec{d}_i \tag{9}$$

where α controls the weight of the feedback vector. Following previous work on PRF [24, 29, 54, 59], we only keep the top t terms with the highest values in the updated query vector $\vec{q}^*$. We then retrieve documents as described in Section 3.5.

3.7 SNRM Summary

In this subsection, we look at the proposed model as a whole and discuss what can actually be learned by the model. From a high-level perspective, SNRM first maps each text to a dense representation in a low-dimensional semantic space and then transforms it to a high-dimensional sparse representation. Based on our retrieval function (see Equation (8)), our model retrieves documents based on the "bag of latent terms" assumption. However, query and document representations are obtained by aggregating ngram representations that capture local term dependencies. Therefore, what SNRM does is mapping the input text from a natural language in which sequences of words matter to a new "latent language" in which term sequences should not play a significant role. Conceptually speaking, SNRM is learning a new "language" with new "vocabulary" terms as its atomic components, and we can benefit from the sparsity of these atomic components in our retrieval framework.

4 EXPERIMENTS

In this section we empirically evaluate and analyze the SNRM. We first introduce the data (Section 4.1), the experimental setup (Section 4.2), and the evaluation (Section 4.3). We then report a range of experimental results and analysis in Section 4.4.

4.1 Data

Collections We evaluate our models using the following two TREC collections: The first collection, Robust, consists of thousands of news articles and is considered as a homogeneous collection. Robust was previously used in TREC 2004 Robust Track. The second collection, ClueWeb, is a challenging and large-scale web collection containing heterogeneous and noisy documents. ClueWeb (i.e., ClueWeb09-Category B) is a common web crawl that only contains English web pages. ClueWeb was previously used in TREC 2009-2012 Web Track. The statistics of these collections as well as the corresponding TREC topics are reported in Table 1. We use the title of topics as queries.

We cleaned the ClueWeb collection by filtering out the spam documents. The spam filtering phase was done using the Waterloo spam scorer[1] [9] with the threshold of 60%. Stopwords were removed from all collections and no stemming was performed.

Training Queries Similar to prior work on weak supervision for information retrieval [16, 52, 53], we generate our weakly labeled data using several million unique queries obtained from the publicly available AOL query logs [38]. This dataset contains a sample of web search queries submitted to the AOL search engine within a three-month period from March 1, 2006 to May 31, 2006. We only used the query strings, and no session and click information was obtained from the query logs. We filtered out the navigational queries containing URL substrings, i.e., "http", "www.", ".com", ".net", ".org", ".edu". All non-alphanumeric characters were removed from the queries. As a sanity check, we made sure that no queries from the training set appear in our evaluation query sets. Applying all of these constraints leads to over 6 million unique queries as our training query set.

[1]http://plg.uwaterloo.ca/~gvcormac/clueweb09spam/

4.2 Experimental Setup

We implemented and trained our models using TensorFlow.[2] The network parameters were optimized with Adam [23] based on the backpropagation algorithm [42]. In our experiments, the learning rate and the batch size were selected from $\{5\times10^{-5}, 1\times10^{-4}, 5\times10^{-4}, 1\times10^{-3}, 5\times10^{-3}\}$ and $\{32,64,128\}$, respectively. Two or three hidden layers were used for the ngram representation network (ϕ_{ngram}) where n is set to 5. The hidden layer sizes were selected from $\{100, 300, 500\}$. The output layer size was also chosen from $\{5k,10k,20k\}$.[3] We select the parameter λ (see Equation (6)) from the $[1\times10^{-7}, 5\times10^{-9}]$ interval. The dropout keep probability was selected from $\{0.6,0.8,1\}$. We initialized the embedding matrix $\mathcal{E}$ by pre-trained GloVe [39] vectors learned from Wikipedia dump 2014 plus Gigawords 5.[4] The embedding dimension was set to 300. All retrieval experiments were carried out using the Galago search engine [10].[5] We performed 2-fold cross-validation over the queries in each collection for tuning the hyper-parameters of our models as well as baselines.

4.3 Evaluation Metrics

To study the effectiveness of SNRM, we report four standard evaluation metrics: mean average precision (MAP) of the top ranked 1000 documents, precision of the top 20 retrieved documents (P@20), normalized discounted cumulative gain [22] calculated for the top 20 retrieved documents (nDCG@20), and recall in the top 1000 documents (Recall). Statistically significant differences of MAP, P@20, nDCG@20, and Recall values were computed using the two-tailed paired t-test with Bonferroni correction at a 95% confidence level.

4.4 Results and Discussion

In this section, we discuss several research questions that are needed to be addressed and for each we present a set of experiments along with their results and analysis to address the research questions.

RQ1: How effective is SNRM compared to the baselines?
To address our first research question, we compare our model against the following baselines:

QL The query likelihood retrieval model [40] with Dirichlet prior smoothing [60]. The smoothing parameter μ is a hyper-parameter selected from $\{100,300,500,1000,1500,2000\}$.

SDM The sequential dependence model of Metzler and Croft [31] that takes advantage of term dependencies using Markov random fields in the language modeling framework. The weight of the unigram query component, the ordered window, and the unordered window were selected from $[0,1]$ with the step size of 0.05, as the hyper-parameters of the model. We made sure that they sum to 1.

RM3 A variant of the relevance model proposed by Lavrenko and Croft [24] that is considered as a state-of-the-art pseudo-relevance feedback model [29]. We selected the number of feedback documents from $\{5,10,15,20,30,50\}$, the feedback term count from $\{10,20,\cdots,100\}$, and the feedback coefficient from $[0,1]$ with the step size of 0.05.

FNRM A pairwise neural ranking model recently proposed by Dehghani et al. [16] (i.e., Rank Model) that uses fully-connected feed-forward networks. This model is based on the bag of words assumption and uses weighted average of word embedding vectors for query and document representations. As suggested in [16], this model re-ranks the top 2000 documents retrieved by query likelihood. Similar to SNRM, this model is trained using the hinge loss in a pairwise setting. The hyper-parameters of this model as listed in the original paper [16] were optimized exactly in the same way as our model (see Section 4.2).

CNRM A neural ranking model based on convolutional networks to take term dependencies into account. In fact, this model uses a convolutional layer on top of the word embedding representations, and then an average pooling layer followed by multiple fully-connected layers is employed. This model is similar to CDSSM [43] and NRM-F [58], except in the use of word embedding instead of trigram hashing. This model is also trained with the same weak supervision data. Hyper-parameter tuning and training setting of this model are similar to those of FNRM.

To have a fair evaluation, we do not compare our model against supervised approaches that require labeled training data.

Table 2 reports the results for the proposed model against the baselines. According to the results, neural ranking models trained with query likelihood as the weak supervision signal (i.e., FNRM, CNRM, and SNRM) significantly outperform the query likelihood model. This indicates that the neural models can generalize their observations from the weak labeler. This happens since the query likelihood model is restricted to term matching and thus suffers from the vocabulary mismatch problem, however, these neural models can do semantic matching learned from a large set of data labeled by QL as the weak labeler. This learning strategy makes it possible to train generalized neural models with no labeled training data. They can potentially improve their weak labelers. This has been also theoretically studied in [53]. The results achieved by SNRM and RM3 are comparable on the Robust collection, while SNRM significantly outperforms RM3 on the ClueWeb collection. Our results also suggest that SNRM performs on par with the FNRM and CNRM baselines, in terms of MAP, P@20, and nDCG@20. It is important to keep in mind that these two baselines are expensive (or even infeasible) to run on large set of documents and thus, as suggested in [16], they only re-rank the top 2000 documents retrieved by query likelihood as the first stage ranker. Therefore, their performances are bounded by the first stage performance, in terms of recall. However, our model, which is able to retrieve documents from its own constructed latent inverted index, can bring up additional relevant documents, even those with no strict vocabulary overlap with the query. It is notable that the Recall@2000 for the QL model is 0.7409 and 0.3551 on Robust and ClueWeb, respectively. Interestingly, the Recall@1000 achieved by SNRM on the Robust collection is higher than the obtained recall by QL on the top 2000 documents. Since FNRM and CNRM are re-ranking the top 2000 documents, our Recall@1000 is even higher than the upper-bound value that can be achieved by any re-ranking baselines, including FNRM and CNRM, on the Robust collection. Note that given the shallow-depth pooling done for assessing the ClueWeb documents, it is not an ideal collection for studying recall-oriented metrics.

It is also worth noting that having a stack of rankers is a common practice in large-scale real-world search engines. Although most existing neural ranking models focus on re-ranking as the final ranker in the stack, our model can be used as an early stage ranker, and improving recall is desirable in such a ranker. Although our learning objective does not directly optimize recall, our model

[2]http://tensorflow.org/
[3]Due to the GPU memory constraints, we used a maximum of 20k dimensions.
[4]https://nlp.stanford.edu/projects/glove/
[5]https://www.lemurproject.org/galago.php

Table 2: Performance of the proposed models and baselines. The highest value per column is marked in bold, and the superscripts 1/2/3/4/5/6 denote statistically significant improvements compared to QL/SDM/RM3/FNRM/CNRM/SNRM, respectively.

Method	Robust				ClueWeb			
	MAP	P@20	nDCG@20	Recall	MAP	P@20	nDCG@20	Recall
QL	0.2499	0.3556	0.4143	0.6820	0.1044	0.3139	0.2294	0.3286
SDM	0.2524	0.3679^{1}	0.4242^{1}	0.6858	0.1078	0.3141	0.2320	0.3385^{1}
RM3	0.2865^{12}	0.3773^{12}	0.4295^{12}	0.7494^{12}	0.1068	0.3157	0.2309	0.3298
FNRM	0.2815^{12}	0.3752^{12}	0.4327^{12}	0.7234^{12}	0.1329^{123}	0.3351^{123}	0.2392^{13}	0.3426^{123}
CNRM	0.2801^{12}	0.3764^{12}	0.4341^{123}	0.7183^{12}	0.1286^{123}	0.3317^{123}	0.2337^{1}	0.3345^{13}
SNRM	0.2856^{12}	0.3766^{12}	0.4310^{12}	0.7481^{1245}	0.1290^{123}	0.3336^{123}	0.2351^{13}	0.3393^{135}
SNRM with PRF	$\mathbf{0.2971}^{123456}$	$\mathbf{0.3948}^{123456}$	$\mathbf{0.4391}^{123456}$	$\mathbf{0.7716}^{123456}$	$\mathbf{0.1475}^{123456}$	$\mathbf{0.3461}^{123456}$	$\mathbf{0.2482}^{123456}$	$\mathbf{0.3618}^{123456}$

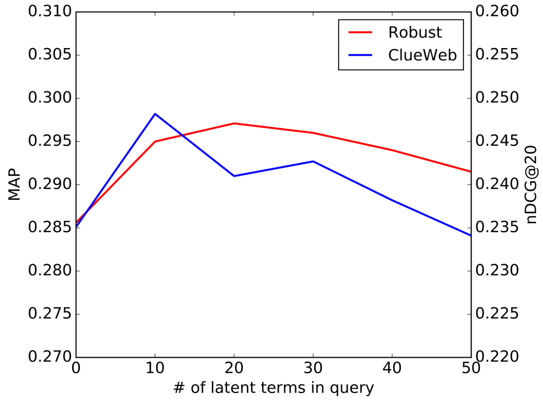

Figure 4: Sensitivity of SNRM with PRF to the number of non-zero elements in the updated query vector.

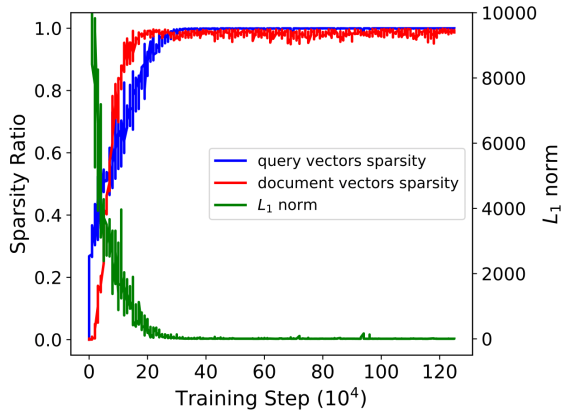

Figure 5: Sparsity ratio for query and document representations plus the L_1 norm with respect to the training steps, for SNRM trained on the Robust collection with 10,000 output dimensionality and $\lambda = 1 \times 10^{-7}$.

improves the baselines in terms of this metric. This indicates that the learned "latent terms" may carry semantic information that is useful for information retrieval purposes.

RQ2: How does pseudo-relevance feedback affect the retrieval performance in the learned latent space?

It is widely known that PRF is effective in many retrieval scenarios [13, 24, 29, 54, 59]. In the next set of experiments, we study the effect of PRF in the new semantic space learned by SNRM. We selected the hyper-parameters α (the feedback weight in Equation (9)), the number of feedback documents, and the number of feedback "terms" (the number of non-zero elements in the updated query vector) using the same cross-validation procedure explained in Section 4.2.

According to Table 2, PRF shows promise in the learned latent space. The reason is that it uses local information obtained from the top retrieved documents. In fact, the top retrieved documents help us to find a better query representation compared to the one obtained by the original short query. SNRM with PRF outperforms all the baselines, including RM3. All the improvements are statistically significant.

The performance of the proposed model with respect to the number of of non-zero elements in the updated query vector (i.e., parameter t) is shown in Figure 4. As suggested by TREC 2004 Robust Track and TREC 2009-2012 Web Track, we use MAP for Robust and nDCG@20 for ClueWeb as the main evaluation metrics. According to Figure 4, the best value for parameter t is 10 for ClueWeb and 20 for Robust, hence dependent on the collection.

RQ3: Does minimizing the L_1 norm promote sparsity in the representations learned by SNRM?.

Increasing the sparsity in the leaned representations is one of the objectives of the model and we translate this into minimizing the L_1 norm. In order to study whether minimizing the L_1 norm promotes sparsity in the representations, we plot the L_1 norm of the learned representations, as well as the sparsity ratio for the input query and documents with respect to the training steps. The definition of sparsity ratio is given in Equation (4). In this experiment, we set the output dimensionality to 10k and the parameter λ (see Equation (6)) to 1×10^{-7}. The results for the model trained on the Robust collection are plotted in Figure 5. These curves show that decreasing in the L_1 norm increases the sparsity in both query and document representations. Besides this observation, it is also shown in Figure 5 that the sparsity in query representations is higher compared to the document representations.

RQ4: How efficient is SNRM at query time?

As mentioned earlier in Section 1, the efficiency brought by term matching models comes from the use of inverted index, which is made possible by the sparsity nature of natural languages. Figure 1 shows that similar to the natural languages, our learned representations also drawn from a Zipfian-like distribution.[6] In addition,

[6]The small sample shown in Figure 1 doesn't exhibit the Zipfian distribution exactly, but it shows the skewed nature that makes the use of an efficient inverted index possible.

Table 3: Number of non-zero elements in the query and document representations with 10,000 output dimensionality.

# Unique latent terms...	Robust		ClueWeb	
	Mean	Std. dev.	Mean	Std. dev.
per document	97.96	447.57	130.24	561.53
per query	3.37	3.04	3.87	4.51

Table 4: Efficiency of SNRM compared to query likelihood, in terms of average run time (milliseconds) per query.

Method	Robust		ClueWeb	
	Mean	Std. dev.	Mean	Std. dev.
QL	35.14	18.43	662.86	746.68
SNRM	46.12	23.11	612.73	640.98

Table 3 reports the number of non-zero elements (i.e., the number of unique latent terms) in the representations learned for queries and documents of Robust and ClueWeb. According to the results, the learned query vectors are much sparser than the document vectors. This shows that the input length affects the sparsity of the learned vectors, which is necessary for an efficient retrieval. The sparsity ratio in Robust is higher than that in ClueWeb. This is due to the document length (see Table 1 for the statistics of the collections) and also the diversity of the documents.

Although the shown properties of the learned representations guarantee an efficient use of inverted index for retrieval, we also study the retrieval time for the proposed model compared to a simple term matching model. To do so, we construct a Galago index from the learned representations and implement our retrieval function (see Equation (8)) as a retrieval model in Galago. The per query running time for SNRM is computed as the query representation time plus the retrieval time. The query representation time is equal to the running time for pre-processing the query text plus a forward pass through the network from query text to the final sparse representation of the query. The retrieval time is the running time for retrieving documents for the obtained query representation from the constructed Galago index. This experiment was run on a machine with a Core i7-4790 CPU @ 3.60GHz and 32GiB RAM. The average and the standard deviation of retrieval time per query for both Robust and ClueWeb collections are reported in Table 4. According to the table, SNRM performs as efficiently as QL, with clearly sub-second response time on the large ClueWeb collection and much faster on the small Robust collection.

RQ5: How does sparsity affect the retrieval performance?

To study this research question, Figure 6 plots the retrieval performance as well as the sparsity ratio achieved by varying the parameter λ (see Equation (6)). As shown in the plot, when λ is set to 1×10^{-5}, the model only focuses on reducing the sparsity, meaning that the learned vector for some queries and documents become all zero. This results in a poor retrieval performance. On ther other hand, when representations have enough non-zero elements, the retrieval performance of the model is stable. For instance, in this case, the performance achieved by 99% and 91% sparseness ratios are close.

RQ6: How sensitive is the performance of the model to the number of unseen documents?

As claimed in Section 3.4, our approach can be also used to index the documents not seen during the training time. To do so, we randomly removed 1% (over 5k documents) and 5% (over 26k documents) from the Robust collection in two different settings. We then trained SNRM using the obtained collections. Once the training was done, we indexed the whole Robust collection with the trained models. The results are reported in Table 5. According to the results, we do not observe a performance loss when 1% of the documents were omitted from the collection. This indicates the robustness of SNRM in indexing unseen documents, which is a practical point in real-world scenarios where the collections frequently change or new documents are added to the collection (e.g., web). However, 5% document removal leads to significant performance drop. Due to the size of the collection, removing 5% of the documents may result in removing a set of vocabulary terms from the collection, and thus the model cannot learn proper latent representations for the documents containing unseen vocabulary terms. This suggests that in real-world scenarios with dynamic collections, the model should be trained periodically, which is already a common practice.

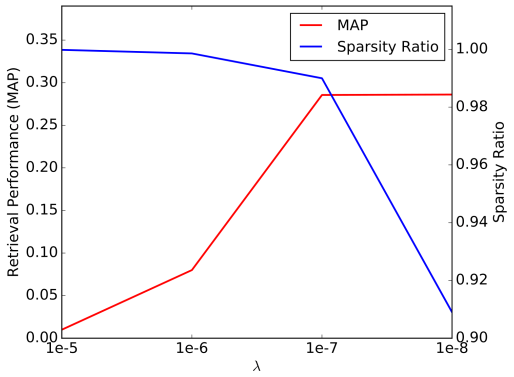

Figure 6: Retrieval performance and sparsity ratio on the Robust collection with respect to different values of parameter λ. The output dimensionality was set to 10,000.

Table 5: Performance of SNRM on the Robust collection with respect to different amount of random document removal at training time. The superscript $^\triangledown$ denotes significant performance loss in comparison with the setting where no document is removed (i.e., no removal).

% removal	MAP	P@20	nDCG@20	Recall
no removal	0.2971	0.3948	0.4391	0.7716
1% removal	0.2953	0.3953	0.4401	0.7691
5% removal	0.2776$^\triangledown$	0.3807$^\triangledown$	0.4227$^\triangledown$	0.7349$^\triangledown$

5 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a *standalone neural ranking model* (SNRM), with an inverted index that results in a small number of short posting lists per query, allowing to retrieve documents from large-scale collections as efficiently as conventional term matching models. Our model learns a high-dimensional sparse representation for queries and documents, optimized for information retrieval. We then construct an inverted index based on the learned sparse representations. At inference or query time, we can efficiently retrieve documents from the entire collection using this inverted index.

We evaluate our models in the context of ad-hoc retrieval using newswire and web collections, and show that our model performs comparably with state-of-the-art neural models that re-rank documents based on the learned dense representations. In addition, we showed that pseudo-relevance feedback is effective in the learned latent space and significantly outperforms competitive baselines.

As future work, we are exploring different ideas to introduce sparsity in the learned representation, beyond minimizing the $L1$-norm, and ways to adapt the loss function to increase the sparsity factor during training. More generally, further analysis of the learned sparse representations is useful to understand which aspects of documents and queries are captured by the "latent terms" in the sparse representation space and to investigate to which extent these aspects capture the notion of relevance for different IR tasks.

Code. An open-source implementation of SNRM is available at https://github.com/hamed-zamani/snrm/.

Acknowledgements. This work was supported in part by the Center for Intelligent Information Retrieval, in part by NSF #IIS-1715095, and in part by NWO #314.99.108. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors.

REFERENCES

[1] M. Aliannejadi, H. Zamani, F. Crestani, and W. B. Croft. Target apps selection: Towards a unified search framework for mobile devices. In *SIGIR '18*, pages 215–224, 2018.
[2] D. Arpit, Y. Zhou, H. Ngo, and V. Govindaraju. Why regularized auto-encoders learn sparse representation? In *ICML'16*, pages 136–144, 2016.
[3] N. Asadi, D. Metzler, T. Elsayed, and J. Lin. Pseudo test collections for learning web search ranking functions. In *SIGIR'11*, pages 1073–1082, 2011.
[4] R. Attar and A. S. Fraenkel. Local feedback in full-text retrieval systems. *J. ACM*, 24(3):397–417, 1977.
[5] L. Azzopardi, M. de Rijke, and K. Balog. Building simulated queries for known-item topics: An analysis using six european languages. In *SIGIR'07*, pages 455–462, 2007.
[6] L. Boytsov, D. Novak, Y. Malkov, and E. Nyberg. Off the beaten path: Let's replace term-based retrieval with k-nn search. In *CIKM'16*, pages 1099–1108, 2016.
[7] Y. Chen and M. J. Zaki. Kate: K-competitive autoencoder for text. In *KDD'17*, pages 85–94, 2017.
[8] D. Cohen, J. Foley, H. Zamani, J. Allan, and W. B. Croft. Universal approximation functions for fast learning to rank: Replacing expensive regression forests with simple feed-forward networks. In *SIGIR '18*, pages 1017–1020, 2018.
[9] G. V. Cormack, M. D. Smucker, and C. L. Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *Inf. Retr.*, 14(5):441–465, 2011.
[10] W. B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. 1st edition, 2009.
[11] M. A. Davenport, M. F. Duarte, Y. C. Eldar, and G. Kutyniok. Introduction to compressed sensing. In *Compressed Sensing: Theory and Applications*, pages 1–64. 2012.
[12] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.
[13] M. Dehghani, H. Azarbonyad, J. Kamps, D. Hiemstra, and M. Marx. Luhn revisited: Significant words language models. In *CIKM '16*, 2016.
[14] M. Dehghani, A. Severyn, S. Rothe, and J. Kamps. Avoiding your teacher's mistakes: Training neural networks with controlled weak supervision. *CoRR*, abs/1711.00313, 2017.
[15] M. Dehghani, A. Severyn, S. Rothe, and J. Kamps. Learning to learn from weak supervision by full supervision. In *Meta-Learning Workshop @ NIPS '17*, 2017.
[16] M. Dehghani, H. Zamani, A. Severyn, J. Kamps, and W. B. Croft. Neural ranking models with weak supervision. In *SIGIR'17*, pages 65–74, 2017.
[17] M. Dehghani, A. Mehrjou, S. Gouws, J. Kamps, and B. Schölkopf. Fidelity-weighted learning. In *ICLR'18*, 2018.
[18] J. Deriu, A. Lucchi, V. De Luca, A. Severyn, S. Müller, M. Cieliebak, T. Hofmann, and M. Jaggi. Leveraging large amounts of weakly supervised data for multi-language sentiment classification. In *WWW'17*, pages 1045–1052, 2017.
[19] D. L. Donoho and B. F. Logan. Signal recovery and the large sieve. *SIAM J. Appl. Math.*, 52(2):577–591, 1992.
[20] J. Guo, Y. Fan, Q. Ai, and W. B. Croft. A deep relevance matching model for ad-hoc retrieval. In *CIKM'16*, pages 55–64, 2016.
[21] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *CIKM'13*, pages 2333–2338, 2013.
[22] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.
[23] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR'15*, 2015.
[24] V. Lavrenko and W. B. Croft. Relevance based language models. In *SIGIR'01*, pages 120–127, 2001.
[25] H. Li. *Learning to Rank for Information Retrieval and Natural Language Processing*. 2011.
[26] Z. Liao, X. Song, Y. Shen, S. Lee, J. Gao, and C. Liao. Deep context modeling for web query entity disambiguation. In *CIKM'17*, pages 1757–1765, 2017.
[27] Z. Lu and H. Li. A deep architecture for matching short texts. In *NIPS'13*, pages 1367–1375, 2013.
[28] C. Luo, Y. Zheng, J. Mao, Y. Liu, M. Zhang, and S. Ma. Training deep ranking model with weak relevance labels. In *ADC'17*, pages 205–216, 2017.
[29] Y. Lv and C. Zhai. A comparative study of methods for estimating query language models with pseudo feedback. In *CIKM'09*, pages 1895–1898, 2009.
[30] A. Makhzani and B. Frey. K-sparse autoencoders. In *ICLR'14*, 2014.
[31] D. Metzler and W. B. Croft. A markov random field model for term dependencies. In *SIGIR'05*, pages 472–479, 2005.
[32] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS'13*, pages 3111–3119, 2013.
[33] B. Mitra and N. Craswell. An introduction to neural information retrieval. *Found. Trends Inf. Retr.*, 2018.
[34] B. Mitra, F. Diaz, and N. Craswell. Learning to match using local and distributed representations of text for web search. In *WWW'17*, pages 1291–1299, 2017.
[35] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(11):2227–2240, 2014.
[36] S. Muthukrishnan. Data streams: Algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, 2005.
[37] Y. Nie, A. Sordoni, and J.-Y. Nie. Multi-level abstraction convolutional model with weak supervision for information retrieval. In *SIGIR '18*, pages 985–988, 2018.
[38] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *InfoScale'06*, 2006.
[39] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP'14*, pages 1532–1543, 2014.
[40] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *SIGIR'98*, pages 275–281, 1998.
[41] J. J. Rocchio. Relevance Feedback in Information Retrieval. In *The SMART Retrieval System: Experiments in Automatic Document Processing*, pages 313–323. 1971.
[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.
[43] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. Learning semantic representations using convolutional neural networks for web search. In *WWW '14*, pages 373–374, 2014.
[44] F. Sun, J. Guo, Y. Lan, J. Xu, and X. Cheng. Sparse word embeddings using l1 regularized online learning. In *IJCAI'16*, pages 2915–2921, 2016.
[45] R. Tibshirani. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Series B*, 58:267–288, 1994.
[46] C. Van Gysel, M. de Rijke, and E. Kanoulas. Learning latent vector spaces for product search. In *CIKM'16*, pages 165–174, 2016.
[47] C. Van Gysel, M. de Rijke, and E. Kanoulas. Neural vector spaces for unsupervised information retrieval. *ACM Trans. Inf. Syst.*, 2018.
[48] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images, Second Edition*. 1999.
[49] C. Xiong, Z. Dai, J. Callan, Z. Liu, and R. Power. End-to-end neural ad-hoc ranking with kernel pooling. In *SIGIR '17*, pages 55–64, 2017.
[50] J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *SIGIR'96*, pages 4–11, 1996.
[51] L. Yu, K. M. Hermann, P. Blunsom, and S. Pulman. Deep learning for answer sentence selection. In *Deep Learning Workshop @ NIPS '14*, 2014.
[52] H. Zamani and W. B. Croft. Relevance-based word embedding. In *SIGIR'17*, pages 505–514, 2017.
[53] H. Zamani and W. B. Croft. On the theory of weak supervision for information retrieval. In *ICTIR'18*, 2018.
[54] H. Zamani, J. Dadashkarimi, A. Shakery, and W. B. Croft. Pseudo-relevance feedback based on matrix factorization. In *CIKM'16*, pages 1483–1492, 2016.
[55] H. Zamani, M. Bendersky, X. Wang, and M. Zhang. Situational context for ranking in personal search. In *WWW'17*, pages 1531–1540, 2017.
[56] H. Zamani, W. B. Croft, and J. S. Culpepper. Neural query performance prediction using weak supervision from multiple signals. In *SIGIR'18*, pages 105–114, 2018.
[57] H. Zamani, M. Dehghani, F. Diaz, H. Li, and N. Craswell. SIGIR 2018 workshop on learning from limited or noisy data for information retrieval. In *SIGIR'18*, pages 1439–1440, 2018.
[58] H. Zamani, B. Mitra, X. Song, N. Craswell, and S. Tiwary. Neural ranking models with multiple document fields. In *WSDM'18*, pages 700–708, 2018.
[59] C. Zhai and J. Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *CIKM'01*, pages 403–410, 2001.
[60] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22(2):179–214, 2004.