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

Hamed Zamani\textsuperscript{(1)}, Mostafa Dehghani\textsuperscript{(2)}, W. Bruce Croft\textsuperscript{(1)}, Erik Learned-Miller\textsuperscript{(1)}, and Jaap Kamps\textsuperscript{(2)}

\textsuperscript{(1)} University of Massachusetts Amherst, USA  
\textsuperscript{(2)} University of Amsterdam, The Netherlands
Motivation

• Learning to rank is core in modern search engines.
  – LTR models (& neural rankers) are stacked re-ranking a small set of documents retrieved by early stage ranker.
  – The performance of LTR & neural models is upper-bounded by the recall of the early stage rankers.

• We propose a **Standalone Neural Ranking Model:**
  – end to end
  – standalone
  – efficient
  – effective
Main Idea

• Looking inside the models:
  – (Effective) models tend to contain many features that aren’t characteristic or essential.
  – They aren’t harmful, so aren’t selected out/removed.
• Can we reward models for being parsimonious?
• Introducing a sparsity property into the learned query and document representations.
  – Building an inverted index for the learned representations.
  – Retrieving documents using the constructed inverted index.
Main Idea

- **term distribution**
- **latent dense representation**
- **latent sparse representation**

The graph shows the document frequency for different representations as a function of the dimension of the representation space.
Objectives

• **Two objectives:**
  
  – **Relevance objective**
    
    • **Hinge loss** as a pairwise loss function
  
  – **Sparsity objective**
    
    • sparsity ratio ($\vec{v}$) = \( \frac{\text{# of zero elements in } \vec{v}}{|\vec{v}|} \)
    
    • Maximizing sparsity ratio is equivalent to **minimizing** \( L_0 \) norm:
      \[
      L_0(\vec{v}) = \sum_{i=1}^{|\vec{v}|} |\vec{v}_i|^0. \text{ (defining } 0^0 = 0) \]
    
    • However, the \( L_0 \) norm is **not differentiable**!
    
    • \( L_1 \) can be used to approximate \( L_0 \).
    
    • Note that we use **ReLU** (\( ReLU(x) = \max(0, x) \)) as the activation function in our network, which sets all non-positive values to zero.
Loss Function

\[ L_{hinge}(q_i, d_{i1}, d_{i2}, y_i) + \lambda \]

A pairwise ranking loss

\[ L_1(q_i | d_{i1} | d_{i2}) \]

concatenation of query and document representations

\[ L_1 \text{ as the sparsity loss} \]

- \( \lambda \) controls the sparsity of the learned representations.

- Note that the representation dimensionality should be high (e.g., up to 20k in our experiments).
Training in a Nutshell

The relevance objective helps the model distinguish relevant vs. non-relevant.

The sparsity objective applies here to increase the sparsity ratio of the final representations.
Post-Training in a Nutshell

How to compute query and document representations?

An offline process for inverted index construction

Query processing
Query and Document Representations

• In designing the representation learning sub-network, we should consider that ...
  – queries generally contain less information that documents. Therefore, query representations are expected to have less non-zero elements.
  – **GPU memory** is limited (e.g., 12GB) and the output dimensionality should be high. The network parameters and the data for each batch should fit into memory.
  – Queries and documents should be in the same semantic space. Therefore, parameters should be **shared**.

• Based on these points, we decided to learn representation for each **n-gram** and aggregate them.
• The network starts with an embedding layer.
  – The embeddings are learned in an **end-to-end** fashion.

• $\phi_{ngram}$ first learns a low dimensional manifold of the data (for high abstraction) and then increases the dimensionality.
Query and Document Representations

\[ \hat{d} = \frac{1}{|d| - n + 1} \sum_{i=1}^{\lfloor \frac{|d|-n+1}{|q|-n+1} \rfloor} \phi_{ngram}(w_i, w_{i+1}, ..., w_{i+n-1}) \]

- \( \phi_{ngram} \) learns a representation for each n-gram.
- Query representation is also computed, similarly.

- Why will this lead to higher sparsity for queries compared to documents?
This model has millions of parameters. How to obtain enough data for training?

Answer: weak supervision.
• Using **Programmatically generated labels**
• Weak supervision for ranking task.
  – Collect large number of queries
  – Using **an existing retrieval model** (e.g., query likelihood), retrieve documents for each query from a collection C.
  – **Assume** that the query likelihood score is true label (!) and train your pairwise model based on this data.

Experiments

Two collections: **Robust** (250 queries) and **ClueWeb09-Cat.B** (200 queries)

Two fold cross-validation over the queries of each collection for hyper-parameter tuning.

**Over 6M unique queries** extracted from **AOL query logs** (excluding navigational queries) for generating weak supervision data using **query likelihood**.

**Metrics:** MAP@1000, P@20, nDCG@20, and Recall@1000
Minimizing $L_1$ leads to higher sparsity ratio.

The number of non-zero elements in query representations is much less than that in the document representations.

The SNRM’s retrieval time (computing query representation + scoring documents using the inverted index) is comparable with term matching models. Therefore, SNRM is efficient.
FNRM [Dehghani et al., SIGIR ‘17] and CNRM (FNRM with convolution) re-rank 2000 documents retrieved by query likelihood.

SNRM outperforms FNRM and CNRM in terms of recall@1000.

SNRM with PRF significantly outperforms all the baselines.
- Note that it is not clear how to incorporate PRF in existing neural models and simple/intuitive approaches do not work well (because of the dense representations)
Robustness to Collection Growth

The retrieval performance by removing some random documents from the Robust collection at the training time.

<table>
<thead>
<tr>
<th>% removal</th>
<th>MAP</th>
<th>P@20</th>
<th>nDCG@20</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>no removal</td>
<td>0.2971</td>
<td>0.3948</td>
<td>0.4391</td>
<td>0.7716</td>
</tr>
<tr>
<td>1% removal</td>
<td>0.2953</td>
<td>0.3953</td>
<td>0.4401</td>
<td>0.7691</td>
</tr>
<tr>
<td>5% removal</td>
<td>0.2776 $\searrow$</td>
<td>0.3807 $\searrow$</td>
<td>0.4227 $\searrow$</td>
<td>0.7349 $\searrow$</td>
</tr>
</tbody>
</table>

- Removing 1% of documents (over 5k documents) from the training set does not significantly affect the performance.
- Removing 5% of documents (over 26k documents) from the training set significantly drops the performance.
  - In the settings where new documents are frequently added to the collection, the model should be re-trained periodically.
Conclusion

• We proposed a **Standalone Neural Ranking Model (SNRM)** that can retrieve documents from a large collection.

• SNRM **does not** require a first stage ranker.
  – It actually can be a **first stage ranker**. It can be potentially trained to maximize recall.

• SNRM is trained **end to end**.

• SNRM can take advantage of **pseudo-relevance feedback**.

• SNRM **outperforms** state-of-the-art IR models.
Thank You!

code: https://github.com/hamed-zamani/snrm