# Neural Ranking Models with Weak Supervision

Dehghani, M.; Zamani, H.; Severyn, A. ; Kamps, J.; Croft, W.B.

*Published in:*
SIGIR'17 : proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval

*DOI:*
10.1145/3077136.3080832

# Neural Ranking Models with Weak Supervision

Mostafa Dehghani*
University of Amsterdam
dehghani@uva.nl

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

Aliaksei Severyn
Google Research
severyn@google.com

Jaap Kamps
University of Amsterdam
kamps@uva.nl

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

## ABSTRACT

Despite the impressive improvements achieved by *unsupervised* deep neural networks in computer vision and NLP tasks, such improvements have not yet been observed in ranking for information retrieval. The reason may be the complexity of the ranking problem, as it is not obvious how to learn from queries and documents when no supervised signal is available. Hence, in this paper, we propose to train a neural ranking model using *weak supervision*, where labels are obtained automatically without human annotators or any external resources (e.g., click data). To this aim, we use the output of an unsupervised ranking model, such as BM25, as a weak supervision signal. We further train a set of simple yet effective ranking models based on feed-forward neural networks. We study their effectiveness under various learning scenarios (point-wise and pair-wise models) and using different input representations (i.e., from encoding query-document pairs into dense/sparse vectors to using word embedding representation). We train our networks using tens of millions of training instances and evaluate it on two standard collections: a homogeneous news collection (Robust) and a heterogeneous large-scale web collection (ClueWeb). Our experiments indicate that employing proper objective functions and letting the networks to learn the input representation based on weakly supervised data leads to impressive performance, with over 13% and 35% MAP improvements over the BM25 model on the Robust and the ClueWeb collections. Our findings also suggest that supervised neural ranking models can greatly benefit from pre-training on large amounts of weakly labeled data that can be easily obtained from unsupervised IR models.

**KEYWORDS** Ranking model; weak supervision; deep neural network; deep learning; ad-hoc retrieval

## 1 INTRODUCTION

Learning state-of-the-art deep neural network models requires a large amounts of labeled data, which is not always readily available and can be expensive to obtain. To circumvent the lack of human-labeled training examples, unsupervised learning methods aim to model the underlying data distribution, thus learning powerful feature representations of the input data, which can be helpful for building more accurate discriminative models especially when little or even no supervised data is available.

A large group of unsupervised neural models seeks to exploit the implicit internal structure of the input data, which in turn requires customized formulation of the training objective (loss function), targeted network architectures and often non-trivial training setups. For example in NLP, various methods for learning distributed word representations, e.g., word2vec [27], GloVe [31], and sentence representations, e.g., paragraph vectors [23] and skip-thought [22] have been shown very useful to pre-train word embeddings that are then used for other tasks such as sentence classification, sentiment analysis, etc. Other generative approaches such as language modeling in NLP, and, more recently, various flavors of auto-encoders [2] and generative adversarial networks [13] in computer vision have shown a promise in building more accurate models.

Despite the advances in computer vision, speech recognition, and NLP tasks using unsupervised deep neural networks, such advances have not been observed in core information retrieval (IR) problems, such as ranking. A plausible explanation is the complexity of the ranking problem in IR, in the sense that it is not obvious how to learn a ranking model from queries and documents when no supervision in form of the relevance information is available. To overcome this issue, in this paper, we propose to leverage large amounts of unsupervised data to infer "noisy" or "weak" labels and use that signal for learning supervised models as if we had the ground truth labels. In particular, we use classic unsupervised IR models as a *weak supervision* signal for training deep neural ranking models. Weak supervision here refers to a learning approach that creates its own training data by heuristically retrieving documents for a large query set. This training data is created automatically, and thus it is possible to generate billions of training instances with almost no cost.[1] As training deep neural networks is an exceptionally data hungry process, the idea of pre-training on massive amount of weakly supervised data and then fine-tuning the model using a small amount of supervised data could improve the performance [11].

The main aim of this paper is *to study the impact of weak supervision on neural ranking models*, which we break down into the following concrete research questions:

**RQ1** *Can labels from an unsupervised IR model such as BM25 be used as weak supervision signal to train an effective neural ranker?*

**RQ2** *What input representation and learning objective is most suitable for learning in such a setting?*

*Work done while interning at Google Research.



*SIGIR '17, August 07-11, 2017, Shinjuku, Tokyo, Japan*
 978-1-4503-5022-8/17/08.
DOI: 10.1145/3077136.3080832

[1] Although weak supervision may refer to using noisy data, in this paper, we assume that no external information, e.g., click-through data, is available.

**RQ3** *Can a supervised learning model benefit from a weak supervision step, especially in cases when labeled data is limited?*

We examine various neural ranking models with different ranking architectures and objectives, i.e., point-wise and pair-wise, as well as different input representations, from encoding query-document pairs into dense/sparse vectors to learning query/document embedding representations. The models are trained on billions of training examples that are annotated by BM25, as the weak supervision signal. Interestingly, we observe that using just training data that are annotated by BM25 as the weak annotator, we can outperform BM25 itself on the test data. Based on our analysis, the achieved performance is generally indebted to three main factors: First, defining an objective function that aims to learn the ranking instead of calibrated scoring to relax the network from fitting to the imperfections in the weakly supervised training data. Second, letting the neural networks learn optimal query/document representations instead of feeding them with a representation based on predefined features. This is a key requirement to maximize the benefits from deep learning models with weak supervision as it enables them to generalize better. Third and last, the weak supervision setting makes it possible to train the network on a massive amount of training data.

We further thoroughly analyse the behavior of models to understand what they learn, what is the relationship among different models, and how much training data is needed to go beyond the weak supervision signal. We also study if employing deep neural networks may help in different situations. Finally, we examine the scenario of using the network trained on a weak supervision signal as a pre-training step. We demonstrate that, in the ranking problem, the performance of deep neural networks trained on a limited amount of supervised data significantly improves when they are initialized from a model pre-trained on weakly labeled data.

Our results have broad impact as the proposal to use unsupervised traditional methods as weak supervision signals is applicable to variety of IR tasks, such as filtering or classification, without the need for supervised data. More generally, our approach unifies the classic IR models with currently emerging data-driven approaches in an elegant way.

## 2 RELATED WORK

Deep neural networks have shown impressive performance in many computer vision, natural language processing, and speech recognition tasks [24]. Recently, several attempts have been made to study deep neural networks in IR applications, which can be generally partitioned into two categories [29, 46]. The first category includes approaches that use the results of trained (deep) neural networks in order to improve the performance in IR applications. Among these, distributed word representations or embeddings [27, 31] have attracted a lot of attention. Word embedding vectors have been applied to term re-weighting in IR models [32, 47], query expansion [10, 33, 43], query classification [25, 44], etc. The main shortcoming of most of the approaches in this category is that the objective of the trained neural network differs from the objective of these tasks. For instance, the word embedding vectors proposed in [27, 31] are trained based on term proximity in a large corpus, which is different from the objective in most IR tasks. To overcome this issue, some approaches try to learn representations in an end-to-end neural model for learning a specific task like entity ranking for expert finding [39] or product search [38]. Zamani and Croft [45] recently proposed relevance-based word embedding models for learning word representations based on the objectives that matter for IR applications.

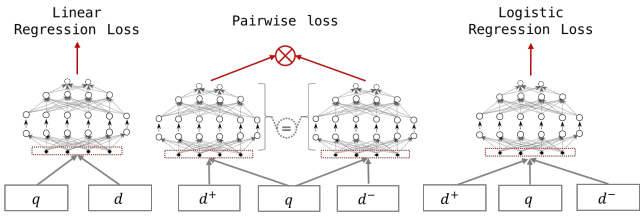
The second category, which this paper belongs to, consists of the approaches that design and train a (deep) neural network for a specific task, e.g., question answering [6, 41], click models [4], context-aware ranking [42], etc. A number of the approaches in this category have been proposed for ranking documents in response to a given query. These approaches can be generally divided into two groups: *late combination models* and *early combination models* (or representation-focused and interaction-focused models according to [14]). The late combination models, following the idea of Siamese networks [5], independently learn a representation for each query and candidate document and then calculate the similarity between the two estimated representations via a similarity function. For example, Huang et al. [18] proposed DSSM, which is a feed forward neural network with a word hashing phase as the first layer to predict the click probability given a query string and a document title. The DSSM model was further improved by incorporating convolutional neural networks [35].

On the other hand, the early combination models are designed based on the interactions between the query and the candidate document as the input of network. For instance, DeepMatch [26] maps each text to a sequence of terms and trains a feed-forward network for computing the matching score. The deep relevance matching model for ad-hoc retrieval [14] is another example of an early combination model that feeds a neural network with the histogram-based features representing interactions between the query and document. Early combining enables the model to have an opportunity to capture various interactions between query and document(s), while with late combination approach, the model has only the chance of isolated observation of input elements. Recently, Mitra et al. [28] proposed to simultaneously learn local and distributional representations, which are early and late combination models respectively, to capture both exact term matching and semantic term matching.

Until now, all the proposed neural models for ranking are trained on either explicit relevance judgements or clickthrough logs. However, a massive amount of such training data is not always available.

In this paper, we propose to train neural ranking models using weak supervision, which is the most natural way to reuse the existing supervised learning models where the imperfect labels are treated as the ground truth. The basic assumption is that we can cheaply obtain labels (that are of lower quality than human-provided labels) by expressing the prior knowledge we have about the task at hand by specifying a set of heuristics, adapting existing ground truth data for a different but related task (this is often referred to distant supervision[2]), extracting supervision signal from external knowledge-bases or ontologies, crowd-sourcing partial annotations that are cheaper to get, etc. Weak supervision is a natural way to benefit from unsupervised data and it has been applied in NLP for various tasks including relation extraction [3, 15], knowledge-base completion [17], sentiment analysis [34], etc. There are also similar attempts in IR for automatically constructing test collections [1] and learning to rank using labeled features, i.e. features that an expert believes they are correlated with relevance [9]. In this paper, we make use of traditional IR models as the weak supervision signal to generate a large

[2]We do not distinguish between weak and distant supervision as the difference is subtle and both terms are often used interchangeably in the literature.




**(a) Score model** **(b) Rank model** **(c) RankProb model**

**Figure 1: Different Ranking Architectures**

amount of training data and train effective neural ranking models that outperform the baseline methods by a significant margin.

## 3 WEAK SUPERVISION FOR RANKING

Deep learning techniques have taken off in many fields, as they automate the onerous task of input representation and feature engineering. On the other hand, the more the neural networks become deep and complex, the more it is crucial for them to be trained on massive amounts of training data. In many applications, rich annotations are costly to obtain and task-specific training data is now a critical bottleneck. Hence, unsupervised learning is considered as a long standing goal for several applications. However, in a number of information retrieval tasks, such as ranking, it is not obvious how to train a model with large numbers of queries and documents with no relevance signal. To address this problem in an unsupervised fashion, we use the idea of "Pseudo-Labeling" by taking advantage of existing unsupervised methods for creating a weakly annotated set of training data and we propose to train a neural retrieval model with weak supervision signals we have generated. In general, weak supervision refers to learning from training data in which the labels are imprecise. In this paper, we refer to weak supervision as a learning approach that automatically creates its own training data using an existing unsupervised approach, which differs from imprecise data coming from external observations (e.g., click-through data) or noisy human-labeled data.

We focus on query-dependent ranking as a core IR task. To this aim, we take a well-performing existing unsupervised retrieval model, such as BM25. This model plays the role of "pseudo-labeler" in our learning scenario. In more detail, given a target collection and a large set of training queries (without relevance judgments), we make use of the pseudo-labeler to rank/score the documents for each query in the training query set. Note that we can generate as much as training data as we need with almost no cost. The goal is to train a ranking model given the scores/ranking generated by the pseudo-labeler as a weak supervision signal.

In the following section, we formally present a set of neural network-based ranking models that can leverage the given weak supervision signal in order to learn accurate representations and ranking for the ad-hoc retrieval task.

## 4 NEURAL RANKING MODELS

In this section, we first introduce our ranking models. Then, we describe the architecture of the base neural network model shared by different ranking models. Finally, we discuss the three input layer architectures used in our neural rankers to encode (query, candidate document) pairs.

### 4.1 Ranking Architectures

We define three different ranking models: one point-wise and two pair-wise models. We introduce the architecture of these models and explain how we train them using weak supervision signals.

**Score model** : This architecture models a point-wise ranking model that learns to predict retrieval scores for query-document pairs. More formally, the goal in this architecture is to learn a *scoring function* $\mathcal{S}(q,d;\theta)$ that determines the retrieval score of document $d$ for query $q$, given a set of model parameters $\theta$. In the training stage, we are given a training set comprising of training instances each a triple $\tau = (q,d,s_{q,d})$, where $q$ is a query from training query set $Q$, $d$ represents a retrieved document for the query $q$, and $s_{q,d}$ is the relevance score (calculated by a weak supervisor), which is acquired using a retrieval scoring function in our setup. We consider the mean squared error as the loss function for a given batch of training instances:

$$\mathcal{L}(b;\theta) = \frac{1}{|b|}\sum_{i=1}^{|b|}(\mathcal{S}(\{q,d\}_i;\theta) - s_{\{q,d\}_i})^2 \tag{1}$$

where $\{q,d\}_i$ denotes the query and the corresponding retrieved document in the $i^{th}$ training instance, i.e. $\tau_i$ in the batch $b$. The conceptual architecture of the model is illustrated in Figure 1a.

**Rank model** : In this model, similar to the previous one, the goal is to learn a scoring function $\mathcal{S}(q,d;\theta)$ for a given pair of query $q$ and document $d$ with the set of model parameters $\theta$. However, unlike the previous model, we do not aim to learn a calibrated scoring function. In this model, as it is depicted in Figure 1b, we use a pair-wise scenario during training in which we have two point-wise networks that share parameters and we update their parameters to minimize a pair-wise loss. In this model, each training instance has five elements: $\tau = (q,d_1,d_2,s_{q,d_1},s_{q,d_2})$. During the inference, we treat the trained model as a point-wise scoring function to score query-document pairs.

We have tried different pair-wise loss functions and empirically found that the model learned based on the hinge loss (max-margin loss function) performs better than the others. Hinge loss is a linear loss that penalizes examples that violate the margin constraint. It is widely used in various learning to rank algorithms, such as Ranking SVM [16]. The hinge loss function for a batch of training instances is defined as follows:

$$\mathcal{L}(b;\theta) = \frac{1}{|b|}\sum_{i=1}^{|b|}\max\Big\{0,\varepsilon - \text{sign}(s_{\{q,d_1\}_i} - s_{\{q,d_2\}_i}) \\ (\mathcal{S}(\{q,d_1\}_i;\theta) - \mathcal{S}(\{q,d_2\}_i;\theta))\Big\}, \tag{2}$$

where $\varepsilon$ is the parameter determining the margin of hinge loss. We found that as we compress the outputs to the range of $[-1,1]$, $\varepsilon = 1$ works well as the margin for the hinge loss function.

**RankProb model** : The third architecture is based on a pair-wise scenario during both training and inference (Figure 1c). This model learns a *ranking function* $\mathcal{R}(q,d_1,d_2;\theta)$ which predicts the probability of document $d_1$ to be ranked higher than $d_2$ given $q$. Similar to the *rank* model, each training instance has five elements: $\tau = (q,d_1,d_2,s_{q,d_1},s_{q,d_1})$. For a given batch of training instances, we

define our loss function based on cross-entropy as follows:

$$\mathcal{L}(b;\theta) = -\frac{1}{|b|}\sum_{i=1}^{|b|} P_{\{q,d_1,d_2\}_i}\log(\mathcal{R}(\{q,d_1,d_2\}_i;\theta)) + (1-P_{\{q,d_1,d_2\}_i})\log(1-\mathcal{R}(\{q,d_1,d_2\}_i;\theta)) \quad (3)$$

where $P_{\{q,d_1,d_2\}_i}$ is the probability of document $d_1$ being ranked higher than $d_2$, based on the scores obtained from training instance $\tau_i$:

$$P_{\{q,d_1,d_2\}_i} = \frac{s_{\{q,d_1\}_i}}{s_{\{q,d_1\}_i} + s_{\{q,d_2\}_i}} \quad (4)$$

It is notable that at inference time, we need a scalar score for each document. Therefore, we need to turn the model's pair-wise predictions into a score per document. To do so, for each document, we calculate the average of predictions against all other candidate documents, which has $O(n^2)$ time complexity and is not practical in real-world applications. There are some approximations could be applicable to decrease the time complexity at inference time [40].

## 4.2 Neural Network Architecture

As shown in Figure 1, all the described ranking architectures share a neural network module. We opted for a simple feed-forward neural network which is composed of: input layer $z_0$, $l-1$ hidden layers, and the output layer $z_l$. The input layer $z_0$ provides a mapping $\psi$ to encode the input query and document(s) into a fixed-length vector. The exact specification of the input representation feature function $\psi$ is given in the next subsection. Each hidden layer $z_i$ is a fully-connected layer that computes the following transformation:

$$z_i = \alpha(W_i.z_{i-1} + b_i);\ 1 < i < l-1, \quad (5)$$

where $W_i$ and $b_i$ respectively denote the weight matrix and the bias term corresponding to the $i^{th}$ hidden layer, and $\alpha(.)$ is the activation function. We use the rectifier linear unit $ReLU(x) = \max(0,x)$ as the activation function, which is a common choice in the deep learning literature [24]. The output layer $z_l$ is a fully-connected layer with a single continuous output. The activation function for the output layer depends on the ranking architecture that we use. For the *score* model architecture, we empirically found that a linear activation function works best, while *tanh* and the sigmoid functions are used for the *rank* model and *rankprob* model respectively.

Furthermore, to prevent feature co-adaptation, we use *dropout* [36] as the regularization technique in all the models. Dropout sets a portion of hidden units to zero during the forward phase when computing the activations which prevents overfitting.

## 4.3 Input Representations

We explore three definitions of the input layer representation $z_0$ captured by a feature function $\psi$ that maps the input into a fixed-size vector which is further fed into the fully connected layers: (i) a conventional dense feature vector representation that contains various statistics describing the input query-document pair, (ii) a sparse vector containing bag-of-words representation, and (iii) bag-of-embeddings averaged with learned weights. These input representations define how much capacity is given to the network to extract discriminative signal from the training data and thus result in different generalization behavior of the networks. It is noteworthy that input representation of the networks in the *score* model and *rank* model is defined for a pair of the query and the document, while the network in the *rankprob* model needs to be fed by a triple of the query, the first document, and the second document.

**Dense vector representation (Dense)** : In this setting, we build a dense feature vector composed of features used by traditional IR methods, e.g., BM25. The goal here is to let the network fit the function described by the BM25 formula when it receives exactly the same inputs. In more detail, our input vector is a concatenation (||) of the following inputs: total number of documents in the collection (i.e., $N$), average length of documents in the collection (i.e., $avg(l_d)_D$), document length (i.e., $l_d$), frequency of each query term $t_i$ in the document (i.e., $tf(t_i,d)$), and document frequency of each query term (i.e., $df(t_i)$). Therefore, for the point-wise setting, we have the following input vector:

$$\psi(q,d) = [N||avg(l_d)_D||l_d||\{df(t_i)||tf(t_i,d)\}_{1\le i\le k}], \quad (6)$$

where $k$ is set to a fixed value (5 in our experiments). We truncate longer queries and do zero padding for shorter queries. For the networks in the *rankprob* model, we consider a similar function with additional elements: the length of the second document and the frequency of query terms in the second document.

**Sparse vector representation (Sparse)** : Next, we move away from a fully featurized representation that contains only aggregated statistics and let the network performs feature extraction for us. In particular, we build a bag-of-words representation by extracting term frequency vectors of query ($tfv_q$), document ($tfv_d$), and the collection ($tfv_c$) and feed the network with concatenation of these three vectors. For the point-wise setting, we have the following input vector:

$$\psi(q,d) = [tfv_c||tfv_q||tfv_d] \quad (7)$$

For the network in *rankprob* model, we have a similar input vector with both $tfv_{d_1}$ and $tfv_{d_2}$. Hence, the size of the input layer is $3 \times vocab\ size$ in the point-wise setting, and $4 \times vocab\ size$ in the pair-wise setting.

**Embedding vector representation (Embed)** : The major weakness of the previous input representation is that words are treated as discrete units, hence prohibiting the network from performing soft matching between semantically similar words in queries and documents. In this input representation paradigm, we rely on word embeddings to obtain more powerful representation of queries and documents that could bridge the lexical chasm. The representation function $\psi$ consists of three components: an embedding function $\mathcal{E} : \mathcal{V} \to \mathbb{R}^m$ (where $\mathcal{V}$ denotes the vocabulary set and $m$ is the embedding dimension), a weighting function $\mathcal{W} : \mathcal{V} \to \mathbb{R}$, and a compositionality function $\odot : (\mathbb{R}^m, \mathbb{R})^n \to \mathbb{R}^m$. More formally, the function $\psi$ for the point-wise setting is defined as:

$$\psi(q,d) = [\odot_{i=1}^{|q|}(\mathcal{E}(t_i^q), \mathcal{W}(t_i^q))||\odot_{i=1}^{|d|}(\mathcal{E}(t_i^d), \mathcal{W}(t_i^d))], \quad (8)$$

where $t_i^q$ and $t_i^d$ denote the $i^{th}$ term in query $q$ and document $d$, respectively. For the network of the *rankprob* model, another similar term is concatenated with the above vector for the second document. The embedding function $\mathcal{E}$ transforms each term to a dense $m$-dimensional float vector as its representation, which is learned during the training phase. The weighting function $\mathcal{W}$ assigns a weight to each term in the vocabulary set, which is supposed to learn term global importance for the retrieval task. The compositionality function $\odot$ projects a set of $n$ embedding and weighting pairs to an

$m$-dimensional representation, independent from the value of $n$. The compositionality function is given by:

$$\odot_{i=1}^{n}(\mathcal{E}(t_i),\mathcal{W}(t_i)) = \sum_{i=1}^{n} \widehat{\mathcal{W}}(t_i)\cdot\mathcal{E}(t_i), \tag{9}$$

which is the weighted element-wise sum of the terms' embedding vectors. $\widehat{\mathcal{W}}$ is the normalized weight that is learned for each term, given as follows:

$$\widehat{\mathcal{W}}(t_i) = \frac{\exp(\mathcal{W}(t_i))}{\sum_{j=1}^{n}\exp(\mathcal{W}(t_j))} \tag{10}$$

All combinations of different ranking architectures and different input representations presented in this section can be considered for developing ranking models.

# 5 EXPERIMENTAL DESIGN

In this section, we describe the train and evaluation data, metrics we report, and detailed experimental setup. Then we discuss the results.

## 5.1 Data

**Collections.** In our experiments, we used two standard TREC collections: The first collection (called *Robust04*) consists of over 500k news articles from different news agencies, that is available in TREC Disks 4 and 5 (excluding Congressional Records). This collection, which was used in TREC Robust Track 2004, is considered as a homogeneous collection, because of the nature and the quality of documents. The second collection (called *ClueWeb*) that we used is ClueWeb09 Category B, a large-scale web collection with over 50 million English documents, which is considered as a heterogeneous collection. This collection has been used in TREC Web Track, for several years. In our experiments with this collection, we filtered out the spam documents using the Waterloo spam scorer[3] [7] with the default threshold 70%. The statistics of these collections are reported in Table 1.

**Training query set.** To train our neural ranking models, we used the unique queries (only the query string) appearing in the AOL query logs [30]. This query set contains web queries initiated by real users in the AOL search engine that were sampled from a three-month period from March 1, 2006 to May 31, 2006. We filtered out a large volume of navigational queries containing URL substrings ("http", "www.", ".com", ".net", ".org", ".edu"). We also removed all non-alphanumeric characters from the queries. We made sure that no queries from the training set appear in our evaluation sets. For each dataset, we took queries that have at least ten hits in the target corpus using the pseudo-labeler method. Applying all these processes, we ended up with 6.15 million queries for the Robust04 dataset and 6.87 million queries for the ClueWeb dataset. In our experiments, we randomly selected 80% of the training queries as training set and the remaining 20% of the queries were chosen as validation set for hyper-parameter tuning. As the "pseudo-labeler" in our training data, we have used BM25 to score/rank documents in the collections given the queries in the training query set.

**Evaluation query sets.** We use the following query sets for evaluation that contain human-labeled judgements: a set of 250 queries (TREC topics 301–450 and 601–700) for the Robust04 collection that were previously used in TREC Robust Track 2004. A set of 200 queries (topics 1-200) were used for the experiments on the ClueWeb collection. These queries were used in TREC Web Track 2009–2012. We only used the title of topics as queries.

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

**Table 1: Collections statistics.**

| Collection | Genre | Queries | # docs | length |
|---|---|---|---|---|
| **Robust04** | news | 301-450,601-700 | 528k | 254 |
| **ClueWeb** | webpages | 1-200 | 50m | 1,506 |

## 5.2 Evaluation Metrics.

To evaluate retrieval effectiveness, we report three standard evaluation metrics: mean average precision (MAP) of the top-ranked 1000 documents, precision of the top 20 retrieved documents (P@20), and normalized discounted cumulative gain (nDCG) [19] calculated for the top 20 retrieved documents (nDCG@20). Statistically significant differences of MAP, P@20, and nDCG@20 values are determined using the two-tailed paired t-test with $p\_value < 0.05$, with Bonferroni correction.

## 5.3 Experimental Setup

All models described in Section 4 are implemented using TensorFlow [12, 37]. In all experiments, the parameters of the network are optimized employing the Adam optimizer [21] and using the computed gradient of the loss to perform the back-propagation algorithm. All model hyper-parameters were tuned on the respective validation set (see Section 5.1 for more detail) using batched GP bandits with an expected improvement acquisition function [8]. For each model, the size of hidden layers and the number of hidden layers were selected from $[16,32,64,128,256,512,1024]$ and $[1,2,3,4]$, respectively. The initial learning rate and the dropout parameter were selected from $[1E{-}3,5E{-}4,1E{-}4,5E{-}5,1E{-}5]$ and $[0.0,0.1,0.2,0.5]$, respectively. For models with embedding vector representation, we considered embedding sizes of $[100,300,500,1000]$. As the training data, we take the top 1000 retrieved documents for each query from training query set $Q$, to prepare the training data. In total, we have $|Q|\times 1000$ ($\sim 6E10$ examples in our data) point-wise example and $\sim |Q|\times 1000^2$ ($\sim 6E13$ examples in our data) pair-wise examples. The batch size in our experiments was selected from $[128,256,512]$. At inference time, for each query, we take the top 2000 retrieved documents using BM25 as candidate documents and re-rank them by the trained models. In our experiments, we use the Indri[4] implementation of BM25 with the default parameters (i.e., $k_1 = 1.2$, $b = 0.75$, and $k_3 = 1000$).

# 6 RESULTS AND DISCUSSION

In the following, we evaluate our neural rankers trained with different learning approaches (Section 4) and different input representations (Section 4.3). We attempt to break down our research questions to several subquestions, and provide empirical answers along with the intuition and analysis behind each question:

**How do the neural models with different training objectives and input representations compare?** Table 2 presents the performance of all model combinations. Interestingly, combinations of the *rank* model and the *rankprob* model with embedding vector representation outperform BM25 by significant margins in both collections. For instance, the *rankprob* model with embedding vector representation that shows the best performance among the other methods,

[4] https://www.lemurproject.org/indri.php

**Table 2: Performance of the different models on different datasets. ▲or ▽indicates that the improvements or degradations with respect to BM25 are statistically significant, at the 0.05 level using the paired two-tailed t-test.**

| Method | Robust04 | | | ClueWeb | | |
|---|---|---|---|---|---|---|
| | *MAP* | *P@20* | *nDCG@20* | *MAP* | *P@20* | *nDCG@20* |
| **BM25** | 0.2503 | 0.3569 | 0.4102 | 0.1021 | 0.2418 | 0.2070 |
| **Score + Dense** | 0.1961▽ | 0.2787▽ | 0.3260▽ | 0.0689▽ | 0.1518▽ | 0.1430▽ |
| **Score + Sparse** | 0.2141▽ | 0.3180▽ | 0.3604▽ | 0.0701▽ | 0.1889▽ | 0.1495▽ |
| **Score + Embed** | 0.2423▽ | 0.3501 | 0.3999 | 0.1002 | 0.2513 | 0.2130 |
| **Rank + Dense** | 0.1940▽ | 0.2830▽ | 0.3317▽ | 0.0622▽ | 0.1516▽ | 0.1383▽ |
| **Rank + Sparse** | 0.2213▽ | 0.3216▽ | 0.3628▽ | 0.0776▽ | 0.1989▽ | 0.1816▽ |
| **Rank + Embed** | **0.2811▲** | **0.3773▲** | **0.4302▲** | **0.1306▲** | **0.2839▲** | **0.2216▲** |
| **RankProb + Dense** | 0.2192▽ | 0.2966▽ | 0.3278▽ | 0.0702▽ | 0.1711▽ | 0.1506▽ |
| **RankProb + Sparse** | 0.2246▽ | 0.3250▽ | 0.3763▽ | 0.0894▽ | 0.2109▽ | 0.1916 |
| **RankProb + Embed** | **0.2837▲** | **0.3802▲** | **0.4389▲** | **0.1387▲** | **0.2967▲** | **0.2330▲** |

surprisingly, improves BM25 by over 13% and 35% in Robust04 and ClueWeb collections respectively, in terms of MAP. Similar improvements can be observed for the other evaluation metrics.

Regarding the modeling architecture, in the *rank* model and the *rankprob* model, compared to the *score* model, we define objective functions that target to learn ranking instead of scoring. This is particularly important in weak supervision, as the scores are imperfect values—using the ranking objective alleviates this issue by forcing the model to learn a preference function rather than reproduce absolute scores. In other words, using the ranking objective instead of learning to predict calibrated scores allows the *rank* model and the *rankprob* model to learn to distinguish between examples whose scores are close. This way, some small amount of noise, which is a common problem in weak supervision, would not perturb the ranking as easily.

Regarding the input representations, embedding vector representation leads to better performance compared to the other ones in all models. Using embedding vector representation not only provides the network with more information, but also lets the network to learn proper representation capturing the needed elements for the next layers with better understanding of the interactions between query and documents. Providing the network with already engineered features would block it from going beyond the weak supervision signal and limit the ability of the models to learn latent features that are unattainable through feature engineering.

Note that although the *rankprob* model is more precise in terms of MAP, the *rank* model is much faster in the inference time ($O(n)$ compared to $O(n^2)$), which is a desirable property in real-life applications.

**Why do dense vector representation and sparse vector representation fail to replicate the performance of BM25?** Although neural networks are capable of approximating arbitrarily complex non-linear functions, we observe that the models with dense vector representation fail to replicate the BM25 performance, while they are given the same feature inputs as the BM25 components (e.g., TF, IDF, average document length, etc). To ensure that the training converges and there is no overfitting, we have looked into the training and validation loss values of different models during the training time. Figure 2 illustrates the loss curves for the training and validation sets (see Section 5.1) per training step for different models. As shown, in models with dense vector representation, the training losses drop quickly to values close to zero while this is not the case for the validation losses, which is an indicator of over-fitting on the training data. Although we have tried different regularization techniques, like $l_2$-regularization and dropout with various parameters, there is less chance for generalization when the networks are fed with the fully featurized input. Note that over-fitting would lead to poor performance, especially in weak supervision scenarios as the network learns to model imperfections from weak annotations. This phenomenon is also the case for models with the sparse vector representation, but with less impact. However, in the models with the embedding vector representation, the networks do not overfit, which helps it to go beyond the weak supervision signals in the training data.

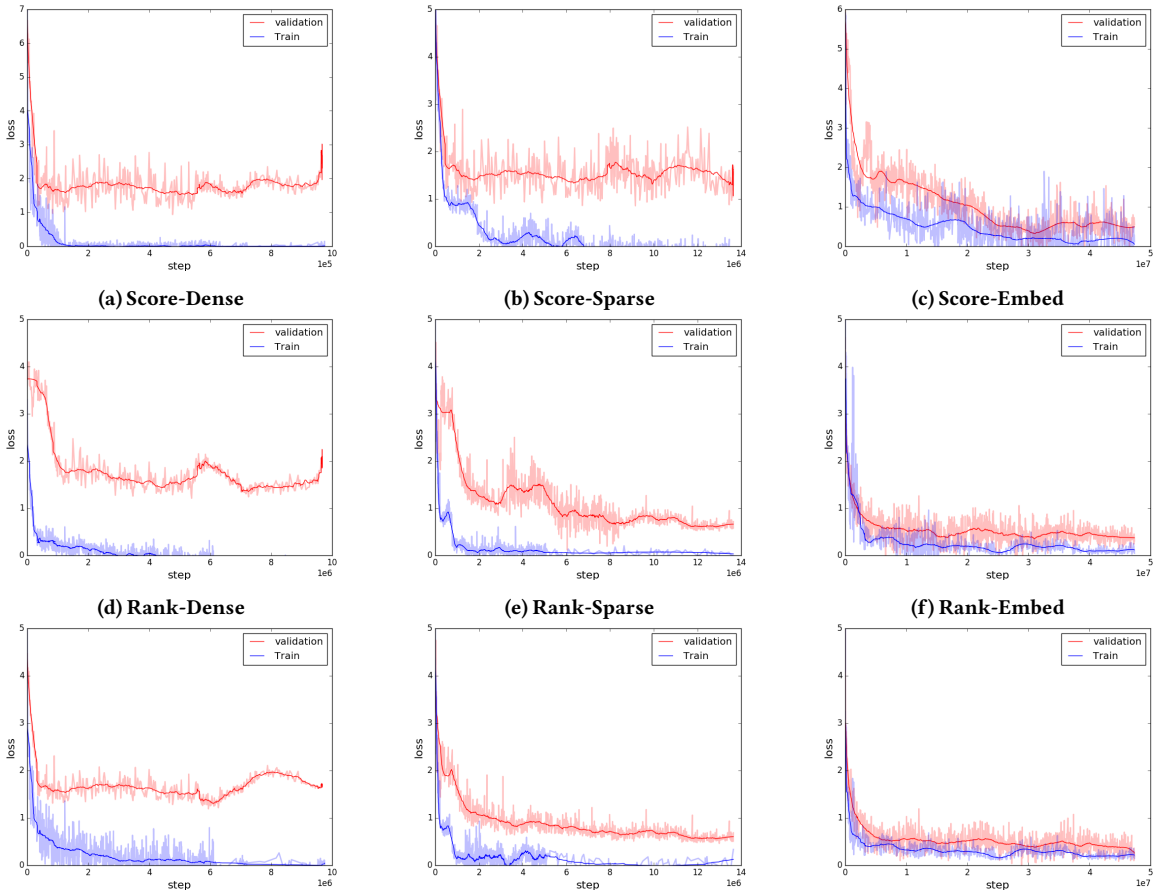
**How are the models related?** To better understand the relationship of different neural models described above, we compare their performance across the query dimension following the approach in [28]. We assume that similar models should perform similarly for the same queries. Hence, we represent each model by a vector, called the performance vector, whose elements correspond to per query performance of the model, in terms of nDCG@20. The closer the performance vectors are, the more similar the models are in terms of query by query performance. For the sake of visualization, we reduce the vectors dimension by projecting them to a two-dimensional space, using t-Distributed Stochastic Neighbor Embedding (t-SNE)[5].

Figure 3 illustrates the proximity of different models in the Robust04 collection. Based on this plot, models with similar input representations (same color) have quite close performance vectors, which means that they perform similarly for same queries. This is not necessarily the case for models with similar architecture (same shape). This suggests that the amount and the way that we provide information to the networks are the key factors in the ranking performance.

We also observe that the *score* model with dense vector representation is the closest to BM25 which is expected. It is also interesting that models with embedding vector representation are placed far away from other models which shows they perform differently compared to the other input representations.

**How meaningful are the compositionality weights learned in the embedding vector representation?** In this experiment, we

[5] https://lvdmaaten.github.io/tsne/




(a) Score-Dense (b) Score-Sparse (c) Score-Embed

(d) Rank-Dense (e) Rank-Sparse (f) Rank-Embed

(g) RankProb-Dense (h) RankProb-Sparse (i) RankProb-Embed

**Figure 2: Training and validation loss curves for all combinations of different ranking architectures and feeding paradigms.**

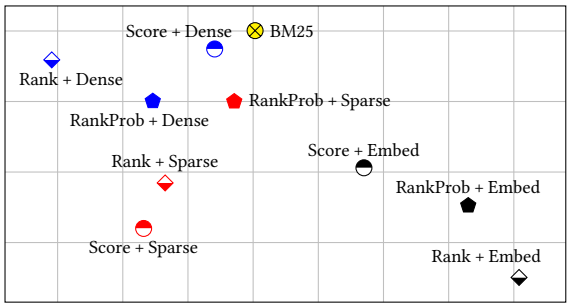
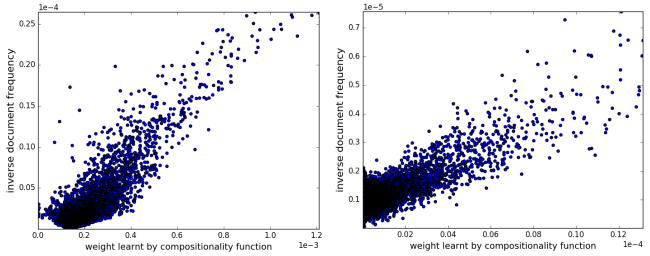



**Figure 3: Proximity of different models in terms of query-by-query performance.**




(a) Robust04 (Pearson Correlation: 0.8243) (b) ClueWeb (Pearson Correlation: 0.7014)

**Figure 4: Strong linear correlation between weight learned by the compositionality function in the embedding vector representation and inverse document frequency.**

focus on the best performing combination, i.e., the *rankprob* model with embedding vector representation. To analyze what the network learns, we look into the weights $\mathcal{W}$ (see Section 4.3) learned by the network. Note that the weighting function $\mathcal{W}$ learns a global weight for each vocabulary term. We notice that in both collections there is a strong linear correlation between the learned weights and the inverse document frequency of terms. Figure 4 illustrates the scatter plots of the learned weight for each vocabulary term and its IDF, in both collections. This is an interesting observation as we do not provide any global corpus information to the network in training and the network is able to infer such a global information by only observing individual training instances.

**How well do other alternatives for the embedding and weighting functions in the embedding vector representation perform?**

Considering embedding vector representation as the input representation, we have examined different alternatives for the embedding

**Table 3: Performance of the *rankprob* model with variants of the embedding vector representation on different datasets. ▲indicates that the improvements over all other models are statistically significant, at the 0.05 level using the paired two-tailed t-test, with Bonferroni correction.**

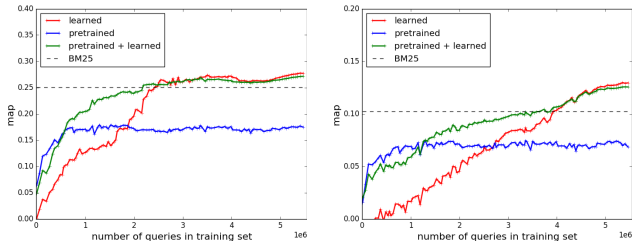
| Embedding type | Robust04 | | | ClueWeb | | |
|---|---|---|---|---|---|---|
| | *MAP* | *P@20* | *nDCG@20* | *MAP* | *P@20* | *nDCG@20* |
| **Pretrained (external) + Uniform weighting** | 0.1656 | 0.2543 | 0.3017 | 0.0612 | 0.1300 | 0.1401 |
| **Pretrained (external) + IDF weighting** | 0.1711 | 0.2755 | 0.3104 | 0.0712 | 0.1346 | 0.1469 |
| **Pretrained (external) + Weight learning** | 0.1880 | 0.2890 | 0.3413 | 0.0756 | 0.1344 | 0.1583 |
| **Pretrained (target) + Uniform weighting** | 0.1217 | 0.2009 | 0.2791 | 0.0679 | 0.1331 | 0.1587 |
| **Pretrained (target) + IDF weighting** | 0.1402 | 0.2230 | 0.2876 | 0.0779 | 0.1674 | 0.1540 |
| **Pretrained (target) + Weight learning** | 0.1477 | 0.2266 | 0.2804 | 0.0816 | 0.1729 | 0.1608 |
| **Learned + Uniform weighting** | 0.2612 | 0.3602 | 0.4180 | 0.0912 | 0.2216 | 0.1841 |
| **Learned + IDF weighting** | 0.2676 | 0.3619 | 0.4200 | 0.1032 | 0.2419 | 0.1922 |
| **Learned + Weight learning** | **0.2837**▲ | **0.3802**▲ | **0.4389**▲ | **0.1387**▲ | **0.2967**▲ | **0.2330**▲ |

function $\mathcal{E}$: (1) employing pre-trained word embeddings learned from an external corpus (we used Google News), (2) employing pre-trained word embeddings learned from the target corpus (using the skip-gram model [27]), and (3) learning embeddings during the network training as it is explained in Section 4.3. Furthermore, for the compositionality function $\odot$, we tried different alternatives: (1) uniform weighting (simple averaging which is a common approach in compositionality function), (2) using IDF as fixed weights instead of learning the weighting function $\mathcal{W}$, and (3) learning weights during the training as described in Section 4.3.

Table 3 presents the performance of all these combinations on both collections. We note that learning both embedding and weighting functions leads to the highest performance in both collections. These improvements are statistically significant. According to the results, regardless of the weighting approach, learning embeddings during training outperforms the models with fixed pre-trained embeddings. This supports the hypothesis that with the embedding vector representation the neural networks learn an embedding that is based on the interactions of query and documents that tends to be tuned better to the corresponding ranking task. Also, regardless of the embedding method, learning weights helps models to get better performance compared to the fixed weightings, with either IDF or uniform weights. Although weight learning can significantly affect the performance, it has less impact than learning embeddings.

Note that in the models with pre-trained word embeddings, employing word embeddings trained on the target collection outperforms those trained on the external corpus in the ClueWeb collection; while this is not the case for the Robust04 collection. The reason could be related to the collection size, since the ClueWeb is approximately 100 times larger than the Robust04.

In addition to the aforementioned experiments, we have also tried initializing the embedding matrix with a pre-trained word embedding trained on the Google News corpus, instead of random initialization. Figure 5 presents the learning curve of the models. According to this figure, the model initialized by a pre-trained embedding performs better than random initialization when a limited amount of training data is available. When enough training data is fed to the network, initializing with pre-trained embedding and random values converge to the same performance. An interesting observation here is that in both collections, these two initializations converge when the models exceed the performance of the weak




**(a) Robust04** **(b) ClueWeb**

**Figure 5: Performance of the *rankprob* model with learned embedding, pre-trained embedding, and learned embedding with pre-trained embedding as initialization, with respect to different amount of training data.**

supervision source, which is BM25 in our experiments. This suggests that the convergence occurs when accurate representations are learned by the networks, regardless of the initialization.

**Are deep neural networks a good choice for learning to rank with weak supervision?** To see if there is a real benefit from using a non-linear neural network in different settings, we examined RankSVM [20] as a strong-performing pair-wise learning to rank method with linear kernel that is fed with different inputs: dense vector representation, sparse vector representation, and embedding vector representation. Considering that off-the-shelf RankSVM is not able to learn embedding representations during training, for embedding vector representation, instead of learning embeddings we use a pre-trained embedding matrix trained on Google News and fixed IDF weights.

The results are reported in Table 4. As BM25 is not a linear function, RankSVM with linear kernel is not able to completely approximate it. However, surprisingly, for both dense vector representation and sparse vector representation, RankSVM works as well as neural networks (see Table 2). Also, compared to the corresponding experiment in Table 3, the performance of the neural network with an external pre-trained embedding and IDF weighting is not considerably better than RankSVM. This shows that having non-linearity in neural networks does not help that much when we do not have representation learning as part of the model. Note that all of these results are still lower than BM25, which shows that they are not good at learning from weak supervision signals for ranking.

**Table 4: Performance of the linear RankSVM with different features.**

| Method | Robust04 | | | ClueWeb | | |
|---|---|---|---|---|---|---|
| | *MAP* | *P@20* | *nDCG@20* | *MAP* | *P@20* | *nDCG@20* |
| **RankSVM + Dense** | 0.1983 | 0.2841 | 0.3375 | 0.0761 | 0.1840 | 0.1637 |
| **RankSVM + Sparse** | 0.2307 | 0.3260 | 0.3794 | 0.0862 | 0.2170 | 0.1939 |
| **RankSVM + (Pretrained (external) + IDF weighting)** | 0.1539 | 0.2121 | 0.1852 | 0.0633 | 0.1572 | 0.1494 |
| **Score (one layer with no nonlinearity) + Embed** | 0.2103 | 0.3986 | 0.3160 | 0.0645 | 0.1421 | 0.1322 |

**Table 5: Performance of the *rankprob* model with embedding vector representation in fully supervised setting, weak supervised setting, and weak supervised plus supervision as fine tuning. ▲indicates that the improvements over all other models are statistically significant, at the 0.05 level using the paired two-tailed t-test, with Bonferroni correction.**

| Method | Robust04 | | | ClueWeb | | |
|---|---|---|---|---|---|---|
| | *MAP* | *P@20* | *nDCG@20* | *MAP* | *P@20* | *nDCG@20* |
| **Weakly supervised** | 0.2837 | 0.3802 | 0.4389 | 0.1387 | 0.2967 | 0.2330 |
| **Fully supervised** | 0.1790 | 0.2863 | 0.3402 | 0.0680 | 0.1425 | 0.1652 |
| **Weakly supervised + Fully supervised** | **0.2912**▲ | **0.4126**▲ | **0.4509**▲ | **0.1520**▲ | **0.3077**▲ | **0.2461**▲ |

We have also examined the *score* model with a network with a single linear hidden layer, with the embedding vector representation, which is equivalent to a linear regression model with the ability of representation learning. Comparing the results of this experiment with Score-Embed in Table 2, we can see that with a single-linear network we are not able to achieve a performance that is as good as a deep neural network with non-linearity. This shows that the most important superiority of deep neural networks over other machine learning methods is their ability to learn an effective representation and take all the interactions between query and document(s) into consideration for approximating an effective ranking/scoring function. This can be achieved when we have a deep enough network with non-linear activations.

**How useful is learning with weak supervision for supervised ranking?** In this set of experiments, we investigate whether employing weak supervision as a pre-training step helps to improve the performance of supervised ranking, when a small amount of training data is available. Table 5 shows the performance of the *rankprob* model with the embedding vector representation in three situations: (1) when it is only trained on weakly supervised data (similar to the previous experiments), (2) when it is only trained on supervised data, i.e., relevance judgments, and (3) when the parameters of the network is pre-trained using the weakly supervised data and then fine tuned using relevance judgments. In all the supervised scenarios, we performed 5-fold cross-validation over the queries of each collection and in each step, we used the TREC relevance judgements of the training set as supervised signal. For each query with $m$ relevant documents, we also randomly sampled $m$ non-relevant documents as negative instances. Binary labels are used in the experiments: 1 for relevant documents and 0 for non-relevant ones.

The results in Table 5 suggest that pre-training the network with a weak supervision signal, significantly improves the performance of supervised ranking. The reason for the poor performance of the supervised model compared to the conventional learning to rank models is that the number of parameters are much larger, hence it needs much more data for training.

In situations when little supervised data is available, it is especially helpful to use unsupervised pre-training which acts as a network pre-conditioning that puts the parameter values in the appropriate range that renders the optimization process more effective for further supervised training [11].

With this experiment, we indicate that the idea of learning from weak supervision signals for neural ranking models, which is presented in this paper, not only enables us to learn neural ranking models when no supervised signal is available, but also has substantial positive effects on the supervised ranking models with limited amount of training data.

## 7 CONCLUSIONS

In this paper, we proposed to use traditional IR models such as BM25 as a weak supervision signal in order to generate large amounts of training data to train effective neural ranking models. We examine various neural ranking models with different ranking architectures and objectives, and different input representations.

We used over six million queries to train our models and evaluated them on Robust04 and ClueWeb 09-Category B collections, in an ad-hoc retrieval setting. The experiments showed that our best performing model significantly outperforms the BM25 model (our weak supervision signal) by over 13% and 35% MAP improvements in the Robust04 and ClueWeb collections, respectively. We also demonstrated that in the case of having a small amount of training data, we can improve the performance of supervised learning by pre-training the network on weakly supervised data.

Based on our results, there are three key ingredients in neural ranking models that lead to good performance with weak supervision: The first is the proper input representation. Providing the network with raw data and letting the network to learn the features that matter, gives the network a chance of learning how to ignore imperfection in the training data. The second ingredient is to target the right goal and define a proper objective function. In the case of having weakly annotated training data, by targeting some explicit labels from the data, we may end up with a model that learned to express the data very well, but is incapable of going beyond it. This is especially the case with deep neural networks where there are many parameters and it is easy to learn a model that overfits the data. The third ingredient is providing the network with a considerable amount of training examples. As an example, during the experiments we noticed that

using the embedding vector representation, the network needs a lot of examples to learn embeddings that are more effective for retrieval compared to pre-trained embeddings. Thanks to weak supervision, we can generate as much training data as we need with almost no cost.

Several future directions can be pursued. An immediate task would be to study the performance of more expressive neural network architectures e.g., CNNs and LSTMs, with weak supervision setup. Other experiment is to leverage multiple weak supervision signals from different sources. For example, we have other unsupervised ranking signals such as query likelihood and PageRank and taking them into consideration might benefit the learning process. Other future work would be to investigate the boosting mechanism for the method we have in this paper. In other words, it would be interesting to study if it is possible to use the trained model on weakly supervised data to annotate data with more quality from original source of annotation and leverage the new data to train a better model. Finally, we can apply this idea to other information retrieval tasks, such as query/document classification and clustering.

## ACKNOWLEDGMENTS

This research was supported in part by Netherlands Organization for Scientific Research through the *Exploratory Political Search* project (ExPoSe, NWO CI # 314.99.108), by the Digging into Data Challenge through the *Digging Into Linked Parliamentary Data* project (DiLiPaD, NWO Digging into Data # 600.006.014), and by the Center for Intelligent Information Retrieval. 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] Nima Asadi, Donald Metzler, Tamer Elsayed, and Jimmy Lin. 2011. Pseudo test collections for learning web search ranking functions. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. ACM, 1073–1082.

[2] Pierre Baldi. 2012. Autoencoders, unsupervised learning, and deep architectures. *ICML unsupervised and transfer learning* 27 (2012), 37–50.

[3] Lidong Bing, Sneha Chaudhari, Richard C Wang, and William W Cohen. 2015. Improving Distant Supervision for Information Extraction Using Label Propagation Through Lists. In *EMNLP '15*. 524–529.

[4] Alexey Borisov, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov. 2016. A Neural Click Model for Web Search. In *WWW '16*. 531–541.

[5] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature Verification Using a "Siamese" Time Delay Neural Network. In *NIPS '93*. 737–744.

[6] Daniel Cohen and W. Bruce Croft. 2016. End to End Long Short Term Memory Networks for Non-Factoid Question Answering. In *ICTIR '16*. 143–146.

[7] Gordon V. Cormack, Mark D. Smucker, and Charles L. Clarke. 2011. Efficient and Effective Spam Filtering and Re-ranking for Large Web Datasets. *Inf. Retr.* 14, 5 (2011), 441–465.

[8] Thomas Desautels, Andreas Krause, and Joel W Burdick. 2014. Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. *Journal of Machine Learning Research* 15, 1 (2014), 3873–3923.

[9] Fernando Diaz. 2016. Learning to Rank with Labeled Features. In *ICTIR '16*. 41–44.

[10] Fernando Diaz, Bhaskar Mitra, and Nick Craswell. 2016. Query Expansion with Locally-Trained Word Embeddings. In *ACL '16*.

[11] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. 2010. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research* 11 (2010), 625–660.

[12] Martín Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/ Software available from tensorflow.org.

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *NIPS*. 2672–2680.

[14] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. A Deep Relevance Matching Model for Ad-hoc Retrieval. In *CIKM '16*. 55–64.

[15] Xianpei Han and Le Sun. 2016. Global Distant Supervision for Relation Extraction. In *AAAI'16*. 2950–2956.

[16] Ralf Herbrich, Thore Graepel, and Klaus Obermayer. 1999. Support Vector Learning for Ordinal Regression. In *ICANN '99*. 97–102.

[17] Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S. Weld. 2011. Knowledge-based Weak Supervision for Information Extraction of Overlapping Relations. In *ACL '11*. 541–550.

[18] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data. In *CIKM '13*. 2333–2338.

[19] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated Gain-based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.* 20, 4 (Oct. 2002), 422–446.

[20] Thorsten Joachims. 2002. Optimizing Search Engines Using Clickthrough Data. In *KDD '02*. 133–142.

[21] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[22] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *NIPS '15*. 3294–3302.

[23] Quoc V Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *ICML '14*, Vol. 14. 1188–1196.

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature* 521, 7553 (2015), 436–444.

[25] Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-yi Wang. 2015. Representation Learning Using Multi-Task Deep Neural Networks for Semantic Classification and Information Retrieval. In *NAACL '15*. 912–921.

[26] Zhengdong Lu and Hang Li. 2013. A Deep Architecture for Matching Short Texts. In *NIPS '13*. 1367–1375.

[27] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS '13*. 3111–3119.

[28] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to Match Using Local and Distributed Representations of Text for Web Search. In *WWW '17*. 1291–1299.

[29] Kezban Dilek Onal, Ismail Sengor Altingovde, Pinar Karagoz, and Maarten de Rijke. 2016. Getting Started with Neural Models for Semantic Matching in Web Search. *arXiv preprint arXiv:1611.03305* (2016).

[30] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. 2006. A Picture of Search. In *InfoScale '06*.

[31] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *EMNLP '14*. 1532–1543.

[32] Navid Rekabsaz, Mihai Lupu, Allan Hanbury, and Hamed Zamani. 2017. Word Embedding Causes Topic Shifting; Exploit Global Context!. In *SIGIR'17*.

[33] Navid Rekabsaz, Mihai Lupu, Allan Hanbury, and Guido Zuccon. 2016. Generalizing translation models in the probabilistic relevance framework. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 711–720.

[34] Aliaksei Severyn and Alessandro Moschitti. 2015. Twitter Sentiment Analysis with Deep Convolutional Neural Networks. In *SIGIR '15*. 959–962.

[35] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. Learning Semantic Representations Using Convolutional Neural Networks for Web Search. In *WWW '14*. 373–374.

[36] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15, 1 (2014), 1929–1958.

[37] Yuan Tang. 2016. TF.Learn: TensorFlow's High-level Module for Distributed Machine Learning. *arXiv preprint arXiv:1612.04251* (2016).

[38] Christophe Van Gysel, Maarten de Rijke, and Evangelos Kanoulas. 2016. Learning latent vector spaces for product search. In *CIKM'16*. 165–174.

[39] Christophe Van Gysel, Maarten de Rijke, and Marcel Worring. 2016. Unsupervised, efficient and semantic expertise retrieval. In *WWW'16*. 1069–1079.

[40] Fabian L. Wauthier, Michael I. Jordan, and Nebojsa Jojic. 2013. Efficient Ranking from Pairwise Comparisons. In *ICML'13*. 109–117.

[41] Liu Yang, Qingyao Ai, Jiafeng Guo, and W. Bruce Croft. 2016. aNMM: Ranking Short Answer Texts with Attention-Based Neural Matching Model. In *CIKM '16*. 287–296.

[42] Hamed Zamani, Michael Bendersky, Xuanhui Wang, and Mingyang Zhang. 2017. Situational Context for Ranking in Personal Search. In *WWW '17*. 1531–1540.

[43] Hamed Zamani and W. Bruce Croft. 2016. Embedding-based Query Language Models. In *ICTIR '16*. 147–156.

[44] Hamed Zamani and W. Bruce Croft. 2016. Estimating Embedding Vectors for Queries. In *ICTIR '16*. 123–132.

[45] Hamed Zamani and W. Bruce Croft. 2017. Relevance-based Word Embedding. In *SIGIR '17*.

[46] Ye Zhang, Md Mustafizur Rahman, Alex Braylan, Brandon Dang, Heng-Lu Chang, Henna Kim, Quinten McNamara, Aaron Angert, Edward Banner, Vivek Khetan, and others. 2016. Neural Information Retrieval: A Literature Review. *arXiv preprint arXiv:1611.06792* (2016).

[47] Guoqing Zheng and Jamie Callan. 2015. Learning to Reweight Terms with Distributed Representations. In *SIGIR '15*. 575–584.