ABSTRACT

Recently, pre-trained language models such as BERT have been applied to document ranking for information retrieval (IR). These methods usually first pre-train a general language model on an unlabeled large corpus and then conduct ranking-specific fine-tuning on expert-labeled relevance datasets. Though preliminary successes have been observed in a variety of IR tasks, a lot of room still remains for further improvement. Ideally, an IR system would model relevance from a user-system dualism: the user’s view and the system’s view. User’s view judges the relevance based on the activities of “real users” while the system’s view focuses on the relevance signals from the system side, e.g., from the experts or algorithms, etc. [20, 41]. Inspired by the user-system relevance views and the success of pre-trained language models, in this paper we propose a novel ranking framework called Pre-Rank that takes both user’s view and system’s view into consideration, under the pre-training and fine-tuning paradigm. Specifically, to model the user’s view of relevance, Pre-Rank pre-trains the initial query-document representations based on a large-scale user activities data such as the click log. To model the system’s view of relevance, Pre-Rank further fine-tunes the model on expert-labeled relevance data. More importantly, the pre-trained representations, are fine-tuned together with handcrafted learning-to-rank features under a wide and deep network architecture. In this way, Pre-Rank can model the relevance by incorporating the relevant knowledge and signals from both real search users and the IR experts. To verify the effectiveness of Pre-Rank, we showed two implementations by using BERT [13] and SetRank [34] as the underlying ranking model, respectively. Experimental results based on three publicly available benchmarks showed that in both of the implementations, Pre-Rank can respectively outperform the underlying ranking models and achieved state-of-the-art performances. The results demonstrate the effectiveness of Pre-Rank in combining the user-system views of relevance.

KEYWORDS

pre-trained IR model; neural information retrieval

1 INTRODUCTION

Relevance ranking, whose objective is to provide the right ranking order of a list of documents for a given query [26, 27], has played a vital role in the field of information retrieval (IR). Machine learning models, especially deep neural networks [16] have been applied to relevance ranking and many ranking techniques have been developed [18, 43]. One branch of the research formalizes the learning of ranking models as first pre-training a general language model on large-scale unlabeled texts and then fine-tuning on the labeled relevance data. For example, Nogueira and Cho [30] consider the training of a passage ranking model as a downstream task in BERT fine-tuning; Chang et al. [6] propose to pre-train a BERT using the Inverse Cloze Task (ICT) as the objective, which aims to teach the model to predict the removed sentence given a context text; Ma et al. [28] proposes a new task of representative words prediction (ROP) to pre-train a BERT model for the ad-hoc retrieval.

Despite improvements that have been observed in many ranking tasks, existing studies either directly consider the ranking learning as a downstream task under the pre-training framework [30], or simply adapt pre-training objectives. The nature and requirements of relevance ranking are rarely taken into consideration. According to the studies in [20, 41], the relevance of a document to a query can be considered from two views: the user’s view and the system’s view. The user’s view prefers that the relevance assessments should be made by the “real users”. The system’s view also called the algorithm’s view, emphasizes systems processing information objects and matching them with queries. Ideally, an IR model would consider both of these two views when conducting the relevance ranking of documents. Existing studies, however, usually model the relevance from only one of the views.

Inspired by the observations and the recent progress of the pre-trained language models, in this paper we propose modeling both the user’s and the system’s view of relevance under the pre-training and fine-tuning framework, called Pre-Rank. In the pre-training stage, it makes use of the real user’s activities, i.e., the large-scale user click log, rather than the unlabeled text corpus for training. Since the user activities imply the relevance judgments of the query-document pairs from the real users, the pre-trained representations are based on the user’s view of relevance, rather than the general natural language processing (NLP) knowledge. One problem with the pre-trained representations is that the user’s activities are usually noisy and contain strong biases (e.g., position bias, selection bias, etc.). To alleviate the issue, Pre-Rank fine-tunes the pre-trained model parameters with query-document pairs with unbiased labels by the experts. To further enhance the ranking performances and incorporating the expert knowledge, Pre-Rank also extends a wide branch [8] to the pre-trained network, resulting in a wide and deep architecture where the wide branch is responsible to inject the handcrafted learning-to-rank features into the model. Since the labeled query-document pairs and the handcrafted features were created based on the expert’s knowledge on the relevance, the fine-tuning stage naturally adapts the ranking model to reflect the

Corresponding author
Pre-Rank offers several advantages. First of all, compared to existing studies in which the pre-trained models are based on unsupervised text data, Pre-Rank makes use of the click log. The pre-trained representations, therefore, reflect the user’s view on relevance, which makes it easy to conduct the downstream fine-tune on expert-labeled relevance data. Second, the handcrafted learning-to-rank features and expert-labeled query-document pairs reflect the expert knowledge (system’s view) on IR relevance, which is complementary to relevance information from user activities in the pre-training stage. It is believed that modeling the two views simultaneously is helpful, especially when very limited labeled query-document pairs are available for fine-tuning. Third, existing studies have shown that the user clicks contain biased information (e.g., position bias, selection bias, etc.) and therefore may hurt the model performances if being directly used as the training corpus [7]. Pre-Rank provides an effective pre-training and fine-tuning approach to relieving the bias issue.

Pre-Rank is a general framework that can involve several types of neural ranking models as its underlying ranking model. As examples, we present two implementations of Pre-Rank based on the state-of-the-art deep ranking models of BERT and SetRank. In the Pre-Rank implementation with BERT, we start with BERT_BASE and continue the pre-training with binary cross-entropy loss on the click log. In the fine-tuning stage, the extended wide and deep BERT network is fine-tuned with expert-labeled data. In the implementation with SetRank, the model is pre-trained on the click log with list-wise cross-entropy loss. After extending with handcrafted features, the wide and deep SetRank network is fine-tuned with the labeled relevance data.

We conducted experiments to test the effectiveness of Pre-Rank by pre-training on large-scale ORCAS click log [9], and fine-tuning on three publicly available benchmarks of MQ2007, MQ2008 [36], and TREC19 [10]. In our experiments, we found that Pre-Rank can respectively outperform the underlying neural ranking models of BERT and SetRank, and achieved state-of-the-art ranking performances on three benchmarks. The results indicate the effectiveness of introducing click-data and handcrafted features in the pre-training and fine-tuning. The experimental analysis also showed that combining the relevance signals from the user’s view and from the system’s view can help to improve the accuracy of relevance ranking.

2 RELATED WORK

One of the most fundamental problems in information retrieval is how to interpret the concept of relevance. Hjørland [20], Saracevic [41] present "the user’s view" and "the system’s view" of relevance. The user’s view formalize and measure relevance based on "real users". In practice, commercial search engines have collected large-scale user activity data (e.g., click log) which provide implicit relevance assessments from the real users. One benefit of using click log is it "removes the cost to the experts of examining and rating the items." [15]. A large number of studies have been conducted to use the click log in search. For example, Joachims [22] trains the ranking SVM model based on the document preference pairs constructed based on the user clicks. Craswell and Szummer [11] apply a Markov random walk model to a large click log. Relevant documents that have not yet been clicked for that query can be retrieved and ranked effectively. Agichtein et al. [1] construct ranking features based on user feedback. See also [14]. One difficulty of using the activities from real users is that the log data contains much noise and biases. Joachims et al. [23] examined the reliability of the implicit relevance information derived from the click data and found that the clicks are informative but biased, including the position biases, selection bias, etc. Therefore, the user clicks could not be treated as relevant judgments [23, 24, 38]. Directly using the click log will inherit biases which is a key difficulty to efficiently use it [25]. Recently, a large number of methods (e.g., counterfactual learning) have been proposed to remove the bias information in the click log [46, 47].

In contrast, the system’s view of relevance (also called algorithmic relevance) “describes the relationship between the query (terms) and the collection of information objects expressed by the retrieved information object(s)” [3]. As one of the representative approaches, traditional learning-to-rank models [26, 27] represent the query-document pairs with handcrafted features by the experts, and learn the ranking models based on labeled relevance data by the annotators [4, 5, 44, 48]. One difficulty of the approach is the high cost of gathering high quality handcrafted features and relevant labels. In recent years, with the availability of large-scale datasets, deep neural networks have also been applied to IR ranking, called neural information retrieval [29]. The neural information retrieval models can automatically learn the query-document features from the raw data, which bridge the gap between query and document vocabulary. For example, some deep approaches [17, 21, 32] focus on discovering the interactions between query and documents and learn complicated interaction patterns. Both traditional handcrafted features and automatically learned features are crucial for relevance ranking [19].

More recently, pre-trained language representation models have led to significant improvements on many NLP tasks [13, 37, 45]. These models are pre-trained on a great amount of unlabeled data with a large neural architecture. As one of the most representative works, BERT [13] gets the language representation model by pre-training on large-scale unlabeled data on a bidirectional transformer-based model. By adding a simple feedforward classification layer on top of BERT, it can outperform many task-specific architectures on various tasks, including IR ranking. There have been studies [12, 30, 31, 49] applied the pre-trained models to the search tasks, by feeding query-document pair into BERT and compute the relevance score over the multi-layer perception (MLP) layer of “[CLS]” token. State-of-the-art performances have been achieved. In [28], the authors designed a specific pre-training task for IR, called representative words prediction, by sampling a pair of word sets according to the document language model.

The pre-trained models usually employ plain text to learn the initial representations, which are originally designed from NLP tasks and do not match well with the task of relevance ranking. In this paper, we try to use a large-scale user click log to pre-train the initial representations and then fine-tune the model on expert-labeled data. From the user-system view of relevance, the work can
be explained as a study of combining these two views in one model. See also the experimental settings in [12, 35].

3 PRE-RANK: OUR APPROACH
The proposed framework, called Pre-Rank, consists of two stages: pre-training stage and fine-tuning stage, in which the model learns the relevance from user’s and system’s view respectively. The overall structure can be seen in Figure 1.

The goal of the pre-training stage is to make the model incorporate specific knowledge of the search domain. In this stage, we used click log to generate supervision signals, because those data can be massively and cheaply obtained, and can reflect the implicit relevance from the user’s view, although such type of signal has some biases problem. We considered raw text feature in this stage because it is a common feature across different ranking datasets.

With a good initialization of the model, the fine-tuning stage aims to directly optimize the target task. Different from the pre-training stage, the fine-tuning stage is suitable to consider dataset dependent features and use expert-labeled corpus as supervision signals. Though it is costly and time-consuming to construct large-scale handcrafted features and expert-labeled corpus, the clean relevance signals make them valuable to adjust the biases and relieve the issues of the pre-trained models. Specifically, we used the combination of the learned features and handcrafted features to represent the query-document pairs and fine-tune the extended ranking model on the expert-labeled corpus.

3.1 Modeling User’s View of Relevance with Pre-train
In the pre-training stage, it is supposed that we are given a set of search activities from the real users (e.g., large-scale click log):

\[ \mathcal{D}^c = \left\{ (q^c, \mathbf{d}^c = (d_1^c, d_2^c, \ldots, d_m^c), \mathbf{e} = (c_1, c_2, \ldots, c_m)) \right\}, \tag{1} \]

where \( q^c = (w_1, \ldots, w_{|q^c|}) \) is a query inputted by real users, where \( w_i \) denotes the \( i \)-th word in the query. Given the query \( q^c \), an IR system will retrieve a set of documents \( \mathbf{d}^c \) presented to the users, where \( d_i^c \) denotes the \( i \)-th document in \( \mathbf{d}^c \). Also, \( d_i^c = (v_1, \ldots, v_{|d_i^c|}) \) also consists of a list of words and \( v_j \) denotes the \( j \)-th word in the document. Let \( e \) be the list of click signals associated with \( \mathbf{d}^c \) where \( c_i \in \{0, 1\} \) denotes whether the \( i \)-th document is clicked by the user where \( c_i = 1 \) if the user clicked \( d_i^c \) and 0 otherwise.

The pre-training stage aims to learn a ranking model \( \mathcal{M}^{pre} \) using a pre-training algorithm \( \mathcal{A}^{pre} \), based on the click log \( \mathcal{D}^c \):

\[ \mathcal{M}^{pre} \leftarrow \mathcal{A}^{pre}(\mathcal{D}^c). \]

Usually, the model \( \mathcal{M}^{pre} \) contains an encoder layer \( \mathcal{E}^{pre}(-) \) and a prediction layer \( \mathcal{P}^{pre}(-) \). The encoder layer aims to generate a vector that encodes the interactions between query and documents. The prediction layer aims to predict the possibility of user clicks on the retrieved documents given the query.

The training algorithm \( \mathcal{A}^{pre} \) consists of two steps: first optimizes the masked language modeling (MLM) objective and then optimizes the click prediction objective. The MLM objective, proposed in [42] for NLP, aims to build the contextual representations for queries and documents, the loss denotes as \( \mathcal{L}_{\text{MLM}} \). Specifically, MLM masks out some tokens from input and then trains the model to predict the masked ones from the rest of the sentence. MLM has been proven to build good contextual representations in many NLP tasks. Following the existing practices [13], the MLM loss is defined as

\[ \mathcal{L}_{\text{MLM}} = - \sum_{\hat{x} \in m(x)} \log p \left( \hat{x} \mid x_{\sim m(x)} \right), \tag{2} \]

where \( x \) is the input sentences, \( m(x) \) are the randomly masked words from \( x \), \( x_{\sim m(x)} \) represent the rest of words from \( x \), and \( p \) is the prediction probability of the masked word \( \hat{x} \) [13].

The click prediction objective models the probability of whether a user will click on the document from \( \mathbf{d}^c \) when entering query \( q^c \), it reflects the relevance between \( q^c \) and \( \mathbf{d}^c \) from the user’s perspective. The loss denotes as \( \mathcal{L}_{\text{click}} \) aims to minimize the differences between the click predictions and the real user clicks:

\[ \mathcal{L}_{\text{click}} = \sum_{(q^c, \mathbf{d}^c) \in \mathcal{D}^c} \mathcal{P}^{pre}(\mathcal{P}^{pre}(q^c, \mathbf{d}^c)), c, \tag{3} \]

where \( \mathcal{P}^{pre} \) is the query-level loss at pre-training stage.

Through optimizing the pre-training objectives, the pre-trained model \( \mathcal{M}^{pre} \) is able to reflect the implicit relevance between query and documents as well as the contextual interaction between query and documents. Noisy and large-scale are two major properties in this stage. Click-through data contains a lot of noise for the sake of positional bias and explosion bias. The noise in the inputs and supervision signals makes it hard to model the relevance, while large-scale data make it possible to involve relevance related information in the model. In the following fine-tuning stage, the expert-labeled data are used to alleviate these issues.

3.2 Modeling System’s View of Relevance with Fine-tune
In the fine-tuning stage, we are given a set of expert-labeled data \( \mathcal{D}^f \) as the training corpus:

\[ \mathcal{D}^f = \left\{ (q^f, \mathbf{d}^f = (d_1^f, d_2^f, \ldots, d_m^f), l = (l_1, l_2, \ldots, l_m)) \right\}, \tag{4} \]

where \( q^f \) is the given query and \( \mathbf{d}^f \) is the document list associate with query \( q^f \), where \( d_i^f \) is the \( i \)-th document in \( \mathbf{d}^f \). Let be the list of
relevance labels associated with $d^l$ where $l_i \in \{0, 1, \cdots, k\}$ denotes how the $i$-th document relevance to the query-relevance labels. Moreover, $q^l = \{w_{i1}, \cdots, w_{i|q^l|}\}$, $d^l = \{c_{i1}, \cdots, c_{i|d^l|}\}$ representing respectively as the raw text of query and document, $w_i$ and $c_i$ denotes the $i$-th word in $q^l$ and $d^l$.

During the fine-tuning stage, we aim to fine-tune the model $M_{pre}$ using algorithm $A_{fine}$ with the expert-labeled data $D^l$:

$$M_{fine} \leftarrow A_{fine}(D^l, M_{pre}).$$

Similar to the pre-training model, $M_{fine}$ also contains an encoder layer $E_{fine}(\cdot)$ and a prediction layer $P_{fine}(\cdot)$. The encoder layer $E_{fine}$ shares the same structure with $E_{pre}$ and is initialized by the pre-trained parameters in $E_{pre}$. During fine-tuning, the parameters in $E_{fine}$ will be further tuned on the labeled data, takes $q^l$ and $d^l$ as input and the output is denoted as $h_{fine}$. $h_{fine} = E_{fine}(q^l, d^l)$. As for $P_{fine}$, the network structure is an extension of $P_{pre}$, by adding a wide branch for involving the handcrafted learning-to-rank features. The parameters in $P_{fine}$ are randomly initialized and tuned in the fine-tuning stage.

The training algorithm $A_{fine}$ aims to minimize the loss between prediction and human label $l$, to get the final ranking list for labeled data:

$$L_{label} = \sum_{(q^l, d^l, l_i) \in D} p_{fine}\left(h_{fine}(\psi(q^l, d^l)), l_i\right),$$

where $E_{fine}$ is the query-level loss at the fine-tuning stage, $\psi(q^l, d^l)$ outputs the handcrafted learning-to-rank features for the input query and document, and $[\cdot, \cdot]$ denotes the concatenation of the inputted vectors.

During the fine-tuning stage, we use unbiased datasets labeled by experts that model relevance from the system’s view, which can alleviate the noise and bias issues with the click log. Clean but small-scale are the main properties in this stage. We use the original features in the benchmark datasets if it is provided, if not, we handcraft ourselves. With the training algorithm $A_{fine}$, model $M_{fine}$ can predict the explicit relevance between query and documents and also leverage the handcrafted features.

## 4 IMPLEMENTATIONS

Pre-Rank is a general ranking framework. In principle, it can be used to improve different ranking methods, by using the method as its underlying ranking model. In this section, we penetrate into the details of two Pre-Rank implementations which use BERT and SetRank as the underlying ranker, denote as Pre-Rank (BERT) and Pre-Rank (SetRank) respectively.

### 4.1 Implementation with BERT

BERT is a language representation model proposed by Devlin et al. [13]. It pre-trains deep bidirectional representations on BooksCorpus [50] and English Wikipedia by using the pre-training objectives of masked language model (MLM) and next sentence prediction. The first token of every input sequence to BERT is always a special classification token ([CLS]). To fine-tune on downstream tasks, the final hidden vector corresponding to the first input token ([CLS]) is fed into a classification layer to get the final prediction. In this paper, we use the off-the-shelf BERT to conduct the pre-training and fine-tuning. The overall architecture of implementing Pre-Rank with BERT is shown in Figure 2.

#### 4.1.1 Pre-training Stage.

The pre-training stage is shown in Figure 2(a). During this stage, the model takes the click log as input. Since BERT is a pair inputted model, for adaptation, we use $(q^l_i, d^l_i, c_i) \in D^l$ as one training instance. As for the encoder layer $E_{pre}$ in BERT, the input is the concatenation of query tokens and the clicked document tokens, with special delimiting tokens i.e., $[CLS] + q^l_i + [SEP] + d^l_i + [SEP]$. The tokens go through several layers of transformers to get fully interaction between query and document. Finally, the output embedding vector of the [CLS] token, denoted as $h_{pre}^l$, is used as a representation for the entire query-document pair, which is obtained by

$$h_{pre}^l = E_{pre}([CLS] + q^l_i + [SEP] + d^l_i + [SEP]).$$

$h_{pre}^l$ is then feed into the prediction layer $P_{pre}$, which is a multi-layer perceptron (MLP) to predict the possibility of click. Cross-entropy loss is used here as the learning objective. Therefore, Eq. (3) can be written as

$$L_{click} = \sum_{(q^l_i, d^l_i, c_i) \in D^l_{BERT}} CE(P_{pre}(E_{pre}(q^l_i, d^l_i)), c_i),$$

where $CE$ denotes the cross-entropy loss function.

During the pre-training stage, the model is first initialized with the pre-trained BERT’s parameters to leverage the language model, while the prediction layer is learned from scratch. The pre-training stage first optimizes the mask language model loss in Eq. (2), and thereafter optimizes the click loss in Eq. (6).

#### 4.1.2 Fine-tuning Stage.

The fine-tuning stage is shown in Figure 2(b). Similar to that of in the pre-training stage, we use $(q^l_i, d^l_i, l_i) \in D^l$ as one training instance. As for the encoder $E_{fine}$ in $M_{fine}$, we initialize the parameters with the pre-trained encoder $E_{pre}$ from first stage, and fine-tune it with labeled data. Following the pre-training stage, the input is the concatenated query and documents tokens: $[CLS] + q^l_i + [SEP] + d^l_i + [SEP]$. The tokens go through several layers of transformers to get fully interaction, and achieve the representation corresponding to the [CLS] token:

$$h_{fine}^l = E_{fine}([CLS] + q^l_i + [SEP] + d^l_i + [SEP]).$$

The output embedding of the [CLS] token $h_{fine}^l$ is concatenated with the handcrafted learning-to-rank features $\psi(q^l_i, d^l_i)$, and then feed into the predictor layer to predict the possibility of relevance. Please note that to involve the handcrafted features, the predictor becomes a wide and deep model architecture and the extended wide part corresponds to the learning-to-rank features. To learning the parameters, the cross-entropy loss between predictions and human labels is constructed and optimized.

$$L_{label} = \sum_{(q^l_i, d^l_i, l_i) \in D^l_{BERT}} CE\left(p_{fine}\left(h_{fine}^l(\psi(q^l_i, d^l_i)), l_i\right)\right).$$

### 4.2 Implementation with SetRank

In this section, we first introduce SetRank and then describe the details of implementation.
SetRank is a listwise ranking model described by Pang et al. [34]. It is a permutation-invariant ranking model defined on document sets of any size. The architecture of SetRank contains three layers, representation layer, encoding layer and ranking layer. The representation layer generates representations of query-document pairs separately with its original learning-to-rank features and ordinal embeddings. The encoding layer jointly processes the documents with multiple sub-layers of multi-head self-attention blocks (MSAB). The final ranking layer calculates the scores and sorts the documents.

4.2.1 Pre-training Stage. The pre-training stage is shown in Figure 3(a). During the pre-training stage, the model takes the click log $D^c$ as the training data, as denoted in Eq. (1). $M^{pre}$ also contains an encoder layer $E^{pre}$ and a prediction layer $P^{pre}$. As for the encoder layer, we use the contextual query and document presentations generated by BERT. Given a query $q^c$ and its associated document set $d^c = (d^c_1, d^c_2, ..., d^c_N)$, each of the document $d^c_i$ in $d^c$ can be represented as a feature vector, the output embedding of the [CLS] token:

$$
\mathbf{h}^{pre}_i = E^{pre}([CLS] + q^c + [SEP] + d^c_i + [SEP]).
$$

All the $(q^c, d^c_i)$ pair in $d^c$’s representation $\mathbf{h}^{pre}$ form a matrix $H^{pre} = [\mathbf{h}^{pre}_1, \mathbf{h}^{pre}_2, ..., \mathbf{h}^{pre}_m]$, is then feed into the prediction layer $P^{pre}$, the core architecture of SetRank, several layers of a Multi-head self-attention block (MSAB) and a row-wise feed-forward network (rFF) to projects each document representation into one real value as the corresponding click score.

Algorithm $A^{pre}$ aim to minimize the loss between prediction and click. Due to SetRank is a list ranking approach, list wise loss is used here to optimize the parameters. That is, Eq. (3) can be written as:

$$
\mathcal{L}_{\text{click}} = \sum_{(q^c, d^c, c) \in D^c} \text{CE}_{\text{list}}(P^{pre}(E^{pre}(q^c, d^c), c), c),
$$

where $\text{CE}_{\text{list}}$ denotes the list-wise cross-entropy loss function. The encoder $E^{pre}$ is initialized with a pre-trained BERT model to leverage the pre-trained language model, while the parameters in $P^{pre}$ are learned from the scratch.

4.2.2 Fine-tuning Stage. As shown in Figure 3(b), in the fine-tuning stage, for each query $q^l$ a list of documents $d^l = (d^l_1, d^l_2, ..., d^l_N)$ is given, as denoted in Eq. (4). Similar to that of in the pre-training stage, each $(q^l, d^l_i)$ pair is represent with BERT:

$$
\mathbf{h}^{fine}_i = E^{fine}([CLS] + q^l + [SEP] + d^l_i + [SEP]).
$$

All the $(q^l, d^l_i)$ pair in $d^l$’s representation $\mathbf{h}^{fine}$ form a matrix on the contextual embedding vectors $H^{fine} = [\mathbf{h}^{fine}_1, ..., \mathbf{h}^{fine}_m]$. Both $H^{fine}$ and $\Psi$ are inputted as the final result of representation layer: $X = [H^{fine}, \Psi]$, where $[\cdot, \cdot]$ denotes concatenation of $H^{fine}$ and learning-to-rank features $\Psi$. Then $X$ is feed into SetRank’s MSAB blocks and rFF to get the final scores. The ranking can be achieved by sorting the documents according to the scores.

The algorithm $A^{fine}$ aims to minimize the loss between the list prediction and human label $l$ of the query-document list:

$$
\mathcal{L}_{\text{label}} = \sum_{(q^l, d^l, l) \in D^l} \text{CE}_{\text{list}}(P^{fine}([H^{fine}, \Psi], l).
$$

5 EXPERIMENTS
We conducted experiments to verify the effectiveness of Pre-Rank on three publicly available IR benchmarks.

5.1 Experimental Settings
In this section, we describe the experimental settings, includes datasets, baselines, evaluation metrics, and experimental details.
5.1.1 Datasets. In all of the experiments, the ranking models were pre-trained on ORCAS [9], a large-scale publicly available click log corpus. ORCAS contains the real user clicks related to the documents used in the TREC Deep Learning Track. The whole ORCAS corpus contains 1.4 million TREC DL URLs, 18 million connections to 10 million distinct queries after the aggregation and filtering (for satisfying the k-anonymity requirements).

The pre-trained models were then fine-tuned on downstream retrieval datasets with human annotated relevance labels. We conducted fine-tune on three datasets MQ2007, MQ2008, and TREC2019, which are published in LETOR 4.0 [36] and TREC19 [10]. Table 1 lists some statistics of the three datasets. MQ2007 contains 1,692 queries 65,323 documents and 69,623 expert-labeled query-document pairs. MQ2008 contains 784 queries 14,384 documents and 15,211 expert-labeled query-document pairs. The number of queries in MQ2008 is relatively small, making it insufficient to learn deep learning model. In the experiments, we followed the practices in [33] and combined the training set of MQ2007 with that of MQ2008 while keeping the validation and test sets unchanged. We still denoted the new dataset as MQ2008. MQ2007 and MQ2008 in total contain 69,623 and 84,834 query documents pairs. The two datasets consist of not only the raw query and document texts, but also 46 dimensions handcrafted features for each query-document pair. TREC 2019 Deep Learning Track benchmark is a large-scale ad-hoc retrieval dataset, which served two tasks: document retrieval and passage retrieval. Both tasks have large training sets with human relevance assessments, derived from MS MARCO [2]. The document retrieval task has a corpus of 3.2 million documents with 367 thousand training queries, and 388,464 labeled query-document pairs. Please note TREC19 only provides the relevant documents for each query. In this paper, we conducted experiments on the sub-task of “top-100 reranking” in document retrieval.

5.1.2 Baselines and Evaluation Metrics. Several types of relevance ranking baselines, including the traditional relevance ranking models, learning-to-rank models, the state-of-the-art neural IR models, were selected in the experiments for comparisons:

- **BM25** [40]: a classical and widely used model for relevance ranking;
- **AdaRank** [44]: a traditional learning-to-rank model that aims to directly optimize the performance measure based on boosting;
- **LambdaMart** [5]: a widely used traditional listwise learning-to-rank model based on gradient boosting;
- **BERT** [13]: a state-of-the-art pre-trained language model which uses MLM and next sentence prediction (NSP) to pre-train the contextual language representation and sentence pair representation. The pre-trained representations can be fine-tuned to a number of downstream tasks, including relevance ranking.
- **SetRank** [34]: a permutation-invariant neural IR model which has the ability to model cross-document interactions so as to capture local context information under a query;
- **PROP** [28]: a recently proposed pre-trained IR model that tailored the training object during pre-training. In practice, PROP was pre-trained on Wikipedia and MS MARCO Document Ranking dataset, denoted as PROP_{wiki} and PROP_{MARCO} respectively.

To evaluate the performances of Pre-Rank and the baselines on MQ2007 and MQ2008, we followed the original data partitions
provided by LETOR4.0 and conducted 5-fold cross validation. The average results were reported. As for the evaluation measures, we used the Precision at 10 (P@10), Normalized Discounted Cumulative Gain at 10 (NDCG@10), and Mean Average Precision (MAP) [39].

To evaluate TREC19, we tested the performances of the proposed Pre-Rank and baselines on the dev set. As for the evaluation measures, we used Recall at 10 (Recall@10) and Mean Reciprocal Rank at 10 (MRR@10).

5.1.3 Experimental Details. During the pre-training procedures, to incorporate relevance assessment from the real users into the ranking model, we performed the pre-training on the ORCAS dataset. ORCAS collected large-scale real user queries, documents, and clicks from a search engine. Like the pre-training of BERT, we also performed two tasks on ORCAS dataset, namely mask language model and click prediction. The procedure of the mask language model task is identical to that of the original BERT model. We randomly masked 15% tokens and made the model to “restore” them. As for the click prediction, the positive instances are derived from the click data of ORCAS, and the negative instances are randomly selected from the top 100 documents of the corresponding query if users did not click them. We combined the two tasks for training the pre-trained model by first running the mask language model task and then running the click prediction task.

During the fine-tuning procedures, we used the pre-trained models to re-rank the candidate documents provided by the datasets. The fine-tuning made use of the raw text of queries, documents, and the handcrafted features provided in the dataset. For MQ2007 and MQ2008, the learning rate was tuned between $10^{-5}$ to $2 \times 10^{-5}$. For fine-tuning Pre-Rank(BERT), we randomly selected 20 documents from the datasets for each query. The input was truncated to 256. For fine-tuning Pre-Ran(SetRank), the list sizes were set to 20 and the input of raw texts were also truncated to 256 for fair comparisons. When conducting the tests on the fine-tuned model, all documents are inputted to get the final ranks. For TREC19 dataset, we randomly select 40 documents per query from the top100 retrieval results officially provided by this dataset, and other settings as the same as MQ2007 and MQ2008 datasets.

Pre-Rank utilized handcrafted relevance features at the fine-tuning stage. These features are combined with the pre-trained representations for making accurate relevance predictions. On MQ2007 and MQ2008, we directly used the learning-to-rank features provided by the LETOR4.0 corpus. Each query-document pair is represented as a 46-dimensional real vector. On TREC19, we extracted 21-dimensional features for each query-document pair. These features reflect the information of the query-level, document-level, and the interaction between the query and document, including query length, BM25 score, tf-idf, query-document matching with bi-grams, tri-grams similarity, word2vec similarity, etc.

For the traditional IR models and learning-to-rank models (i.e., BM25, AdaRank, and LambdaMart), we used the implementations shared in [33]. The implementations of BERT and Pre-Rank are based on the popular Transformers library ¹. As for BERT, the encoder layer of Pre-Rank(BERT) and the encoder layer of Pre-Rank(SetRank) are initialized by BERT_BASE’s checkpoint release by Google ².

5.2 Experimental results

Table 2 reports the experimental results of the proposed Pre-Rank and all of the baselines on the downstream datasets of MQ2007, MQ2008, and TREC19. The baselines’ results mainly follow [33]. For PROP_wiki and PROP_MARCO we used the numbers reported in [28] thus have no results on TREC19 and in terms of MAP on MQ2007 and MQ2008. The results on TREC19 of PROP are based on the released models in [28].

By comparing Pre-Rank (BERT) with BERT and Pre-Rank (SetRank) with SetRank, we can see that Pre-Rank outperformed the underlying neural IR model of BERT and SetRank with large margin. We conducted significant tests on the improvements. The results indicate that all of the improvements over the raw underlying ranking models are significant ($p$-value < 0.05). Considering that BERT and SetRank are already very strong neural IR baselines, the results indicate the effectiveness of the Pre-Rank framework. It also verified the effectiveness of Pre-Rank’s approach that involving both the user’s view and the system’s view of relevance signals in one model using the pre-training and fine-tuning paradigm. Comparing the two implementations, we found that Pre-Rank (SetRank) performed better than Pre-Rank (BERT), verified the advantages of the permutation-invariant ranking model.

From the results, we also found that: (1) the neural IR models of BERT and SetRank can obtain better results than traditional models such as BM25, AdaRank and LambdaMART, indicating that automatically learned features can capture more relevance signals other than the handcrafted learning-to-rank features; (2) the pre-trained ranking model PROP [28] improved BERT on MQ2007 and MQ2008 by a huge margin, verified the power of the pre-training and fine-tuning paradigm; (3) the proposed Pre-Rank (SetRank) outperformed PROP in most cases, verified the effectiveness and necessity of modeling the relevance from both user’s view and system’s view.

5.3 Discussions

We conducted experiments to show the reasons that our approaches outperformed the baselines and impacts of different parameter settings.

5.3.1 Impact of handcrafted features at the fine-tuning stage. One of the unique characteristics of Pre-Rank is its ability to involve both the pre-trained representations and handcrafted learning-to-rank features. In the experiments, we tested the effects of the handcrafted features. Specifically, we tested the ranking accuracies of Pre-Rank (BERT) and Pre-Rank (SetRank) when the wide parts (which are responsible for injecting learning-to-rank features) were removed during the fine-tuning. In the experiments, all of the inputs were under the same setting for each dataset. Table 3 lists the experimental results. In the table, the Pre-Rank (BERT) and Pre-Rank (SetRank) versions with and without the handcrafted features were denoted as “w/ ltrFtr” and “w/o ltrFtr”. From the results, we can

¹https://github.com/huggingface/transformers

²https://github.com/google-research/bert
see that in most cases combined with the automatically pre-trained features, the handcrafted features can further improve the ranking accuracy under the Pre-Rank framework. The results verified that though the powerfulness of the pre-trained representations, the expert knowledge encoded in the traditional learning-to-rank features (one aspect of the system’s view on relevance) is still valuable for relevance ranking.

5.3.2 Selection of the learning objectives at the pre-training stage.
During the pre-training stage, Pre-Rank utilized two learning objectives: the traditional mask language model (MLM) objective based on the raw texts, and the click prediction objective based on the real user’s click activities. We conducted experiments to investigate the impacts of these two learning objectives in Pre-Rank. Specifically, we removed one of the objectives and pre-trained the initial representations with the remained objective. The fine-tuning stage is kept unchanged. Table 4 shows the performances of the variations of Pre-Rank (BERT) and Pre-Rank (SetRank) in which the pre-training objectives are adjusted. In the table, we denote the experiments pre-training with MLM objective only, pre-training with click prediction objective only, and pre-training with MLM and click prediction as “w/MLM”, “w/click”, and “w/MLM + Click”, respectively. From the results reported in Table 4, we found that Pre-Rank models pre-trained with click prediction objective only (“w/Click”) performed similarly to the full version: pre-training with both MLM and click prediction objectives (“w/MLM + Click”). On the other hand, the model that pre-trained with MLM objective only (“w/MLM”) performed much worse. The phenomenon can be observed in both the experiments with Pre-Rank (BERT) and Pre-Rank (SetRank). The results clearly indicate that: (1) the activities from the real users (user’s view of relevance) are critical relevance signals for relevance ranking; (2) through existing studies have proven MLM to be an effective pre-training objective for many NLP related downstream tasks, it seems that MLM is not an optimal selection for relevance ranking. Both the user’s view and the system’s view on relevance also rarely touch the NLP knowledge of the texts in the queries and documents, especially the knowledge that can be derived from the mask language modeling. Therefore, it is easy to understand why very limited improvements can be observed after adding the MLM objective to the click prediction objective.

5.3.3 Ability to improve long queries. We also conducted experiments to show on which kinds of queries our approaches can perform well. Specifically, we categorized the queries of MQ2007 into different query groups according to the lengths of the queries. For each query group, we tested the performance improvements of Pre-Rank (BERT) over the underlying ranking model of BERT, in terms of NDCG@10, and the performance improvements of Pre-Rank (SetRank) over the underlying ranking model of SetRank in terms of NDCG@10. Figure 4 showed the NDCG@10 improvements w.r.t. the query groups. The results are the average values over the 5 folds’ test set. We can see that there is a trend that Pre-Rank can achieve more improvements on the query groups with long query lengths (e.g., the group whose query length is 10). This is particularly obvious in the case of Pre-Rank (BERT). One reason for the phenomenon is that by pre-training on raw text from the user’s view, Pre-Rank models more semantic relevance and therefore improves when the query is longer and requires more semantic information.

![Figure 4: NDCG@10 improvements of Pre-Rank (BERT) and Pre-Rank(SetRank) over the underlying ranking models w.r.t. different query lengths on MQ2007.](image-url)
Table 3: Impacts of the handcrafted learning-to-rank features at the fine-tuning stage.

| Settings             | MQ2007 | MQ2008 | TREC19 |
|----------------------|--------|--------|--------|
|                      | P@10  | NDCG@10| MAP    | MRR@10 | Recall@10 |
| Pre-Rank (BERT)      |        |        |        |        |           |
| w/o LtrFtr           | 0.430  | 0.516  | 0.518  | 0.256  | 0.252     | 0.513   | 0.375  | 0.642  |
| w/ LtrFtr            | 0.430  | 0.520  | 0.521  | 0.255  | 0.252     | 0.514   | 0.376  | 0.644  |
| Pre-Rank (SetRank)   |        |        |        |        |           |
| w/o LtrFtr           | 0.433  | 0.521  | 0.520  | 0.256  | 0.252     | 0.510   | 0.389  | 0.643  |
| w/ LtrFtr            | 0.436  | 0.526  | 0.525  | 0.258  | 0.258     | 0.521   | 0.388  | 0.648  |

Table 4: Impacts of the pre-training objectives in Pre-Rank.

| Settings             | MQ2007 | MQ2008 | TREC19 |
|----------------------|--------|--------|--------|
|                      | P@10  | NDCG@10| MAP    | MRR@10 | Recall@10 |
| Pre-Rank(BERT)       |        |        |        |        |           |
| w/MLM                | 0.424  | 0.502  | 0.504  | 0.253  | 0.246     | 0.502   | 0.371  | 0.636  |
| w/MLM+Click          | 0.430  | 0.520  | 0.521  | 0.255  | 0.252     | 0.514   | 0.376  | 0.644  |
| w/Click              | 0.432  | 0.521  | 0.518  | 0.255  | 0.252     | 0.507   | 0.373  | 0.641  |
| Pre-Rank(SetRank)    |        |        |        |        |           |
| w/MLM                | 0.418  | 0.487  | 0.496  | 0.256  | 0.253     | 0.515   | 0.387  | 0.644  |
| w/MLM+Click          | 0.436  | 0.526  | 0.525  | 0.258  | 0.258     | 0.521   | 0.388  | 0.648  |
| w/Click              | 0.436  | 0.526  | 0.525  | 0.259  | 0.258     | 0.523   | 0.387  | 0.645  |

5.3.4 Effects of combining relevance signals from user’s view and system’s view. We conducted experiments to test the effects of combining the user’s view and the system’s view on relevance in learning ranking models. Specifically, we trained SetRank models based on three settings of the training data: (1) trained on the ORCAS dataset that only contains clicks from real users (denoted as “Click-Only”); (2) trained on the TREC19 dataset that only contains relevance labels by the experts (denoted as “Label-Only”); (3) first trained on the ORCAS click log, and then continued the training on TREC19 expert-labeled dataset (denoted as “Click + Label”). We tested the performances on TREC19’s dev set and Figure 5 shows the performance curves of these three settings w.r.t. training steps in terms of MRR@10. From the results, we can conclude that (1) the large-scale click log (relevance signals from the user’s view) is effective in training ranking models in the early stages. However, as the training goes on, the noise and biases in the click log hurt the training procedure and hinder the further improvements; (2) the limited but clean labeled data (relevance signals from the system’s view) is useful in training and the model’s performance improved steadily until convergence; (3) the ranking performances can be further improved when the model trained on ORCAS was continually trained on TREC19 data. This is because these two views are complementary: the clean and unbiased labeled data in TREC19 relieves the noise and bias issues with the click log, while the large-scale click log provides massive relevance signals to overcome the scale limitation.

6 CONCLUSION

This paper proposes a novel relevance ranking framework under the pre-training and fine-tuning paradigm, for modeling the relevance information from both the user’s view and the system’s view. The framework, called Pre-Rank, first pre-trains the initial representations on real user’s search activities (e.g., click log) and then fine-tunes the ranking model on expert-labeled data with handcrafted learning-to-rank features. We implemented two versions of Pre-Rank which used BERT and SetRank as its underlying ranking model, respectively. Experimental results on publicly available benchmarks showed that after pre-training on large-scale user activities data, both of the pre-trained Pre-Rank versions can be well fine-tuned on three benchmarks and significantly enhanced the ranking performances.

ACKNOWLEDGMENTS

This work was funded by the National Key R&D Program of China (2019YFE0198200), the National Natural Science Foundation of China (No. 61872338, No. 61906180, No. 61832017), Beijing Academy of Artificial Intelligence (BAAI2019ZD0305), and Beijing Outstanding Young Scientist Program NO. BJKWZYJH012019100020098.
