Effective and Practical Neural Ranking

A Dissertation
submitted to the Faculty of the
Graduate School of Arts and Sciences
of Georgetown University
in partial fulfillment of the requirements for the
degree of
Doctor of Philosophy
in Computer Science

By

Sean MacAvaney, M.S.

Washington, DC
March 24, 2021

Effective and Practical Neural Ranking

Sean MacAvaney, M.S.

Dissertation Advisors: Nazli Goharian and Ophir Frieder

Abstract

Supervised machine learning methods that use neural networks ("deep learning") have yielded substantial improvements to a multitude of Natural Language Processing (NLP) tasks in the past decade. Improvements to Information Retrieval (IR) tasks, such as ad-hoc search, lagged behind those in similar NLP tasks, despite considerable community efforts. Although there are several contributing factors, I argue in this dissertation that early attempts were not more successful because they did not properly consider the unique characteristics of IR tasks when designing and training ranking models. I first demonstrate this by showing how large-scale datasets containing weak relevance labels can successfully replace training on in-domain collections. This technique improves the variety of queries encountered when training and helps mitigate concerns of over-fitting particular test collections. I then show that dataset statistics available in specific IR tasks can be easily incorporated into neural ranking models alongside the textual features, resulting in more effective ranking models. I also demonstrate that contextualized representations, particularly those from transformer-based language models, considerably improve neural ad-hoc ranking performance. I find that this approach is neither limited to the task of ad-hoc ranking (as demonstrated by ranking clinical reports) nor English content (as shown by training effective cross-lingual neural rankers). These efforts demonstrate that neural approaches can be *effective* for ranking tasks. However, I observe that these techniques are impractical due to their high query-time computational costs. To overcome this, I study approaches for offloading computational cost to index-time, substantially reducing

query-time latency. These techniques make neural methods *practical* for ranking tasks. Finally, I take a deep dive into better understanding the linguistic biases of the methods I propose compared to contemporary and traditional approaches. The findings from this analysis highlight potential pitfalls of recent methods and provide a way to measure progress in this area going forward.

INDEX WORDS: Information retrieval, Deep learning, Efficient ad-hoc search, Natural language processing

Acknowledgments

This dissertation would not have been possible without the support of innumerable colleagues, friends, and family members. First and foremost, I thank my academic co-advisors: Nazli Goharian and Ophir Frieder. They enabled me to pursue my interests while also pushing me to continue to improve and pursue challenging new directions. I also thank the remaining members of my dissertation committee—Andrew Yates, Jimmy Lin, Justin Thaler, Nathan Schneider, and Nicola Tonellotto—who provided constructive and valuable feedback on my proposal and dissertation. Andrew and Nicola deserve additional gratitude for serving as mentors during internships throughout my Ph.D., along with Arman Cohan, Doug Downey, Franck Dernoncourt, Franco Maria Nardini, Raffaele Perego, Sergey Feldman, and Walter Chang. Each of these internships enabled me to pursue new and interesting research directions, with my internship mentors' support. Next, I would like to thank my remaining co-authors and collaborators—Amir Zeldes, Ayah Zirikly, Bart Desmet, Canjia Li, Christian Wolf, Craig Macdonald, Cristopher Flagg, Eugene Yang, Gholam Motamedi, Hao-Ren Yao, Ish Talati, Kai Hui, Katina Russell, Jay Urbain, Joe Garman, Luca Soldaini, Ross Filice, Sajad Sotudeh, Tong Xiang, and Zeus De los Santos—with whom I was able to study a variety of fascinating topics both within and outwith this dissertation.

I would also like to thank the many others I met at Georgetown who were wonderful and supportive friends, including Jakob Prange, Johnson Truong, Jordana Bickel, Mohammad Zaheri, Peter Haferl, and countless others. Finally, I would like to thank those back home who would always cheer me on, including my mother Kathy,

father Paul, sister Kelly, brother Dennis, friends Brandon Nickel, Jenny Carrillo, Kristen Schmalfeldt, Shane Zoltak, Sienna Bast, and many others.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Search engines are ubiquitous, with over a trillion queries processed annually and millions of daily users [1]. Although the exact details of popular search engines remain elusive due to their proprietary nature, at their core, they still rely heavily on lexical keyword matching and document quality estimates (e.g., PageRank [144]) to retrieve and rank results [2, 170]. Lexical ranking is limited, though, since they treat word matches the same regardless of context. Intuitively, a search engine with an improved understanding of a document's textual content should be able to do a better job ranking documents scoring a document that uses a query term in a relevant context above documents that use the term incidentally or in a different sense. Although Natural Language Processing (NLP) techniques such as word sense disambiguation [175] and semantic indexing [200] have been previously applied to search, these attempts were largely unsuccessful. With neural network techniques now being used successfully in NLP, we can now model more complex relationships between words in a given text. This dissertation demonstrates that these neural techniques can effectively rank documents for ad-hoc searches and show how neural ranking can do this with a reasonable impact on query-time latency. I further study the limitations and linguistic biases introduced by these models.

In the past decade, supervised neural machine learning approaches have dominated the field of NLP [142]. The improvements that neural approaches offer NLP

are largely a result of effective understanding of lexical semantics[1] through distributional modeling. This is accomplished via a two-step process. In the first step, the language is modeled via word co-occurrence in massive amounts of text. This stage results in a model that places words with similar semantic meaning (inferred through similar co-occurrences) at similar points in space. In the second step, supervised modeling tunes the original model for a particular task [44, 151]. This stage exploits these similarities to model a particular task by treating semantically-similar words similarly. This process helps overcome the issue of sparsity in language [155], since most tasks have a limited amount of training data. Recent improvements to word representations have allowed the modeling of words in a given textual context, which provides models with the capacity to learn particular word senses and resolve co-references (e.g., ELMo [153] and BERT [44]).

There have been high hopes that these techniques can apply to Information Retrieval (IR), particularly for ad-hoc ranking. In this task, a textual query provided by a user is used to score documents from a large collection, such that documents with content relevant to the query are assigned the highest scores. Dozens of research papers were published on this topic (e.g., [37, 56, 73, 75, 76, 126, 127, 136, 138, 146, 148, 186, 193, 214, 228]), and multiple workshops focusing on this topic were held [32, 33]. Despite these efforts, few had been able to overcome simple (yet well-tuned) lexical baselines, such as BM25 [167] (a commonly-used probabilistic retrieval model from the mid 1990s) [97, 218]. This is surprising because neural approaches should allow for the semantic matching of terms, as they do for other NLP tasks.

Several factors contribute to the apparent lack of effectiveness of neural ranking approaches. For instance, as Lin points out in [97], one contributing factor is the utilization and comparison against weak baselines. There is also some evidence that

[1] I focus on semantics at the word and sub-word level.

when using and incorporating stronger baselines, many of the apparent gains disappear [218]. As I demonstrate in this dissertation, another important factor is inadequate consideration of the training environment and task-specific characteristics. As I show in Section 2.2, the decision to train neural ranking models on limited-scale datasets (which were built for *evaluation* of ranking methods, rather than the large-scale training needed by neural methods) can lead to inferior models. There are also valid concerns that training and evaluating from the same dataset can overfit to the dataset, which results in models that are not generally applicable or exhibit artifacts of the annotation process [57]. Most attempts at neural ranking have also largely failed to incorporate valuable dataset characteristics for modeling, which I demonstrate can be easily incorporated into ranking models (Section 2.3).

The aforementioned work in Chapter 2 predates the prominence of large-scale contextualized language models (such as BERT [44]) in NLP. These models offer a considerable opportunity for neural IR; they allow models to take advantage of massive amounts of unlabeled natural language to help model words as they appear in a given context. In Chapter 3, I demonstrate ways in which these models can be beneficial to IR tasks. Section 3.2 shows that term representations from contextualized language models can be incorporated into existing neural ranking architectures to great benefit. In this process, I also show that signals from the contextualized language model alone can produce a reasonably-effective ranker.[2] In Section 3.3, I show that these signals can be useful for another ranking task, namely the ranking of discrepancies in clinical reports. In Section 3.4 I show that contextualized language models can also benefit ranking for languages that have little or no relevance training data available by transferring relevance signals from larger English collections. In Section 3.5, I loop back to the topic of training data and show how adaptive weighing

[2]Several others made this observation contemporaneously [135, 156, 220].

training samples can improve BERT model effectiveness. And finally, in Section 3.6 I revisit the topic of relevance transfer across tasks first discussed in Section 2.2. I show that it is still an effective technique—given new models and a new large-scale training resource—through experiments on a new evaluation dataset focused on search over COVID-19 literature [166].

Although Chapters 2–3 demonstrate that one can use neural methods effectively for ranking, they overlook an important aspect needed to use these models in practice: computational efficiency. Specifically, inefficient models would be expensive for search engines to run and result in a poor user experience due to increased time waiting for a response. Though there are various ways to measure efficiency, I focus on query-time latency: the total time it takes to score and rank documents for a given query. Indeed, approaches based on contextualized language models incur considerable costs in terms of this measure. By some benchmarks, conventional techniques are over 150× faster than methods using contextualized language models; a query that would have taken 20 milliseconds to execute instead takes over 3 seconds. In Chapter 4, I show that—although effective at ranking—these approaches have a considerable negative impact on query-time latency. I demonstrate techniques to mitigate this cost by offloading computational cost to index time. I first show a general approach that involves partially computing representations (Section 4.2). I then show how specialized ranking architectures can be designed that reduce the computational time even further by building sparse query and document representations (Section 4.3).

Finally, in Chapter 5, I investigate possible biases and unintended side-effects of using these models for ranking. Through probing tasks built to isolate specific model behaviors, I find that models based on contextualized language models exhibit fundamentally different behaviors than conventional approaches. For instance, although lexical signals such as term frequency affect these models' rankings, they play a much

lesser role in scoring; they can distinguish relevance without these signals effectively. But this has consequences. Models unintuitively favor documents that contain extra non-relevant information and slightly favor documents that are more formal or fluent (even when the underlying message is identical), which can potentially harm language learners. These are behaviors not exhibited by traditional models and warrant consideration before deploying a search engine using these methods.

1.1 Hypotheses

I formulate the above into the following concrete hypotheses:

Hypothesis 1: *Neural ranking methods suffer from ineffective training environments, which can be mitigated.*

The first part of my dissertation examines how neural ranking models training and modeling, demonstrating how to improve effectiveness. This is covered in Chapters 2 and 3.

> **Hypothesis 1.1:** *Relevance signals can be transferred across datasets to overcome poor training data.*
>
> In Section 2.2,I show that existing training resources can be inadequate in scale and diversity to build effective ranking models. I show that one can easily overcome this by utilizing sources of *weak supervision*. Specifically, I demonstrate that massive amounts of naturally-occurring text pairs (e.g., headlines and news articles) can effectively replace traditional sources of labeled relevance data. Then, in Section 3.6, I show how relevance signals can be transferred across datasets to a new task when contextualized language models are used.

Hypothesis 1.2: *Utilizing dataset characteristics can improve the effectiveness of neural ranking models.*

Dataset characteristics, such as Inverse Document Frequency (IDF), have long shown to be effective signals in retrieval models, yet neural ranking methods largely overlook them. In Section 2.3, I demonstrate that statistics from the dataset can be effectively incorporated into neural ranking models. Specifically, I show how structured query information can be used to improve the task of complex answer retrieval. Later, in Section 3.5, I show how dataset characteristics can be used to weight samples during training to produce more effective models.

Hypothesis 1.3: *Enhanced text representations improve the effectiveness of ranking models.*

I demonstrate in Section 3.2 that contextualized text representations (e.g., BERT [44]) can replace static word embeddings in neural ad-hoc ranking models, and they substantially improve ranking performance. Additionally, incorporating the pretrained classification mechanism from contextualized language models further improves ranking effectiveness. In Section 3.3, I show that this can be useful for another ranking task: ranking the degree of discrepancy in revisions of clinical reports.

Hypothesis 1.4: *Relevance training can be effectively transferred across natural languages.*

Most training resources for IR are only in English. In Section 3.4, I demonstrate that ranking models capable of effectively ranking documents in multiple languages can be trained using only English relevance pairs. I also show that adding

a small amount of in-language training data can further improve ranking performance.

Hypothesis 2: *Neural rankers can be implemented with low query-time latency.*

Chapter 4 of my dissertation examines the computational efficiency of using neural ranking models. Specifically, I examine the effect that contextualized language models have on query-time latency, which has the most substantial impact on end-users.

Hypothesis 2.1: *Offloading computations to index time reduces query-time latency of neural rankers.*

In Section 4.2, I show that not all calculations necessarily need to be done at query-time and that offloading these computations to index-time can reduce the query-time latency of using these models. This approach can improve the efficiency of any ranker that uses contextualized language models.

Hypothesis 2.2: *Specialized ranking architectures can be designed to reduce query-time latency.*

Armed with the knowledge that not all calculations are necessary to rank documents effectively, I propose a new model that attempts to reduce the query-time burden as much as possible in Section 4.3. This proposed model builds sparse document representations built at index-time and scored at query-time using an inexpensive dot product operation. I find that this model can significantly improve ranking effectiveness in a re-ranking setting while contributing as little as 5ms to the total ranking time.

Hypothesis 3: *Neural rankers exhibit fundamentally different behaviors than lexical ranking methods, which can lead to unintended effects.*

Chapter 5 proposes new probing tasks to understand the behaviors of retrieval models better. Using these probes, I find that rankers based on contextualized language models exhibit linguistic biases not present in lexical models. I find that these biases are non-trivial to overcome without sacrificing the model's ranking effectiveness.

1.2 Organization

Given the fast pace of progress in this area, results in each section are presented as they were at the time of original publication. That is, new baselines were not added retroactively to studies. Particularly influential was shift to large-scale contextualized language models. In this dissertation, I re-validate the hypotheses that I originally validated prior to the rise of contextualized language models, albeit using the newer techniques and resources that are available at the time. For instance, when validating Hypothesis 1.1, I originally demonstrated relevance transfer across datasets using the New York Times corpus [176] and non-contextualized models (Section 2.2). I later re-explore this topic using contextualized language models and the MS-MARCO collection [19] with a revised methodology (Section 3.6).

The remainder of this dissertation is organized as follows. In Chapters 2 and 3, I present support for Hypothesis 1. The introduction of contextualized language models for ranking (e.g., BERT [44]) represents a major shift in the field, and thus approaches employed prior to *contextualization*[3] are presented in Chapter 2 and those after contextualization in Chapter 3. In Chapter 4, I shift focus towards model efficiency and provide support for Hypothesis 2. Finally, I present support for Hypothesis 3 in

[3] I refer to the usage of neural contextualized language models for ranking as *contextualization*, for short, even though prior efforts to introduce context into neural ranking models exist (e.g., [76]).

Chapter 5 through an extensive analysis of model behaviors. In Chapter 6, I draw final conclusions of my dissertation and place the work in a broader context.

Parts of Chapters 2–5 are reproductions of my jointly authored publications [108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119].

Chapter 2

Ranking Effectiveness of Neural Models without Contextualization

There has been considerable effort in the IR community to design neural network architectures capable of ranking documents for ad-hoc queries effectively [32, 33]. Although seemingly effective, many of these approaches only appeared to improve ad-hoc ranking due to comparisons against weak baselines [97]. In this chapter, I demonstrate that these efforts were nevertheless worthwhile by showing that (1) these neural ranking architectures be trained effectively using weak supervision labels (i.e., their utility extends beyond a single dataset), and (2) that these approaches can easily incorporate task-specific dataset statistics, which can further improve ranking effectiveness.

The remainder of this chapter is organized as follows. Section 2.1 sets the stage by going over prior work in neural ranking. Section 2.2 presents work on content-based weak supervision for neural ranking, addressing Hypothesis 1.1. Section 2.3 details work on one particular task, complex answer retrieval, and shows how dataset characteristics can easily be incorporated into neural ranking models, addressing Hypothesis 1.2. Section 2.4 briefly summarizes the innovations described in this chapter.

2.1 Background and Preliminaries

The task of ad-hoc ranking involves sorting a collection of textual documents by relevance to a particular user query. Let $\mathbf{q} = \{q_1, q_2, ..., q_n\}$ be a query consisting of

n tokens, and $\mathbf{d} = \{d_1, d_2, ..., d_m\}$ a document consisting of m tokens. Let D be a collection of documents. The task of ad-hoc ranking is to order the documents in D such that the documents that are most relevant to the given query are positioned near the top of the list $r_{\mathbf{q},D} = [\mathbf{d}_1, \mathbf{d}_2, ..., \mathbf{d}_{|D|}]$. This is accomplished by assigning a relevance score $rel(q, d)$ for every $d \in D$, and ordering the documents in descending order by the score.

Prior to the recent advances in neural information processing, there was a considerable amount of work done to predict document relevance scores based on simple term occurrence statistics (*lexical* models). Popular lexical models have included vector space modeling, probabilistic modeling, and language modeling [55]. The two that are commonly used as baseline methods are Okapi BM25 [81] and the Query Likelihood model [154]. Others have developed so-called learning-to-rank approaches, which make use of training data to combine simple features (e.g., inverse document frequency). The most notable learning-to-rank approach is LambdaMART [17], which builds a tree ensemble over engineered features such as TF-IDF scores, PageRank [144], spam scores, etc. Neural ranking approaches are similar in that they are supervised approaches, but instead relying on engineered features, they operate over the query and document content itself. Thus, they are able to directly learn what type of content is important to match and the types of patterns are indicative of relevance. These models also may incorporate engineered features (e.g., it is common to include IDF scores), but the main focus is to learn how to encode and process the query and document content itself.

One can generalize most neural ranking architectures to $rel(\mathbf{q}, \mathbf{d}) = F(\Phi_q(\mathbf{q}), \Phi_d(\mathbf{d}))$. $\Phi.(\cdot)$ is the *text encoding* function, which encodes the query or document into a particular representation. Note that $\Phi_q(\cdot)$ and $\Phi_d(\cdot)$ are often (but not necessarily) the same function. $F(\cdot)$ is the *matching combination* function which compares the two

encodings to produce a final relevance score. This notation can be used demonstrate the difference between the two major categories of neural matching architectures proposed by [146]: representation-focused and interaction-focused. *Representation-focused* architectures (Section 2.1.3) employ a Siamese network, where $\Phi(\cdot)$ is a deep neural network yielding a dense representation for a given document, and $F(\cdot)$ is usually a simple similarity measure (e.g., cosine similarity). In contrast, *interaction-focused* architectures (Section 2.1.3) use a simple $\Phi(\cdot)$ (e.g., the identify function), mapping the query and document to their corresponding sequence of terms. Here, $F(\cdot)$ is a deep neural network that determines textual matches over the interaction matrix of the query and document.

2.1.1 Training and Inference

Based on traditional learning-to-rank techniques, systems can be trained to either optimize for item-wise loss, pair-wise loss, or list-wise loss. Most work in neural information retrieval has rallied around pair-wise loss, which tends to work better than the others in practice. Specifically, during training a query $\mathbf{q}$ is selected along with a known relevant document $\mathbf{d}^+$ and non-relevant document $\mathbf{d}^-$ for that query. In practice, $\mathbf{d}^+$ simply needs to be *more* relevant than $\mathbf{d}^-$. This means that for graded relevance, $\mathbf{d}^+$ could be considered *highly relevant*, while $\mathbf{d}^-$ could be *possibly relevant*. Training in this way improves graded relevance scores like nDCG.

There are two common forms of pair-wise loss functions. The first is hinge loss, employed by several systems [56, 75]:

$$\mathcal{L}(\mathbf{q}, \mathbf{d}^+, \mathbf{d}^-; \Theta) = \max(0, 1 - rel(\mathbf{q}, \mathbf{d}^+) + rel(\mathbf{q}, \mathbf{d}^-)) \tag{2.1}$$

where Θ are the model parameters. Hinge loss has recently fallen out of vogue, as cross-entropy loss has been shown to be more effective [43, 76]:

$$\mathcal{L}(\mathbf{q}, \mathbf{d}^+, \mathbf{d}^-; \Theta) = -\log \frac{\exp(rel(\mathbf{q}, \mathbf{d}^+))}{\exp(rel(\mathbf{q}, \mathbf{d}^+)) + \exp(rel(\mathbf{q}, \mathbf{d}^-))} \tag{2.2}$$

During training, random training samples are selected and fed through the network, optimizing for the loss in mini-batches for a fixed number of training epochs. It is common practice to use a validation set to select the optimal trained model from among the epochs using a validation metric such as Mean Average Precision (MAP) [76].

Most training datasets can be characterized as being either deep or wide. Deep datasets, such as TREC Robust 2004 [201], have human-labeled relevance judgments for a large proportion of the document collection for each query. However, the cost of building such judgments typically results in only a small number of queries. Deep datasets are beneficial for evaluation due to a low proportion of unjudged documents present in the top results. Wide datasets have a large number of queries, but only a few relevance judgments per query (either manually-labeled from top-scoring documents, e.g., MS-MARCO [132], or through a labeling heuristic, e.g., CAR [45]). Datasets based on user interaction data (such as those based on query logs) are also considered wide, but such datasets are usually unavailable to researchers due to their proprietary nature. The variety and scale of these datasets make them attractive for training models, but can be considered insufficient for evaluation because of their high proportion of unjudged documents for each query.

Some have also proposed approaches for neural pseudo-relevance feedback (NPRF) [94]. In this strategy, the top-ranked documents by an unsupervised ranker (e.g., BM25) are fed back into the model as additional query terms. For practical reasons, only the top terms by IDF in the pseudo-relevant documents are considered.

This means that the approach provides less benefit to (and perhaps harms) models that use n-gram matching (e.g., ConvKNRM [37] and PACRR [76], see Section 2.1.3).

At inference time, using neural rankers is straightforward. Given a ranking of documents $r^k_{\mathbf{q},D}$ for document $\mathbf{q}$ for collection D at cutoff k, each document is assigned the relevance score from the model: $rel(\mathbf{q}, \mathbf{d})$ for each $\mathbf{d} \in r^k_{\mathbf{q},D}$. The use of the cutoff k is a practical matter: D can be very large (can contain billions of documents in web search[1]) which would take a considerable amount of time to run, and the user will often be able to fully satisfy their information need from at most several documents. However, the use of a cutoff can remove potentially relevant documents that do not meet the criteria of the initial ranking. This approach is considered a *telescoping* approach (also also referred to as cascading, multi-staged, pipelined, re-ranking) [123, 207]. In this dissertation, I focus on using a simple ranking function, such as BM25, to produce the initial candidate list.

Others have proposed doing completely end-to-end neural ranking, by learning sparse or dense representations [84, 215, 228]. In practice, performing re-ranking atop these results are often still beneficial. Others have explored using neural networks to add expansion terms at query-time [136, 138].

Neural rankers are often evaluated on standard benchmark datasets (covered below), or on proprietary query log information. Note that proprietary datasets make the replication of results difficult. With recent developments in large-scale datasets for neural ranking, this problem should hopefully be reduced.

2.1.2 Ranking Datasets

Several ad-hoc retrieval datasets exist, and can be used for both the training and evaluation of neural ranking approaches. The following are the datasets most com-

[1] `http://lemurproject.org/clueweb09.php/`

Table 2.1 Comparison of datasets commonly used for evaluating neural ad-hoc retrieval performance.

			Relevance Judgments		
Dataset	Documents	Topics	Total	Per topic	All rel.
TREC Robust 2004	$528k$	249	$311k$	$1.2k$	
TREC WebTrack 2009–12	$1.0B$	200	$84k$	420	
TREC WebTrack 2013–14	$733M$	100	$28k$	280	
TREC CAR (automatic)	$30M$	$2.2M$	$5.2M$	2.3	✓
TREC CAR (manual)	$30M$	702	$30k$	23	
MS-MARCO (train)	$8.8M$	$809k$	$533k$	0.7	✓

The 'All rel' column indicates that all the collection contains only positive relevance judgments (no non-relevance judgments).

monly used for neural information retrieval. Some were designed with data-hungry neural approaches in mind, whereas others have other desirable qualities which make them appealing for training or evaluating these approaches. A summary is provided in Table 2.1.

TREC Robust 2004 [201]. This dataset contains a document collection of $528k$ and 249 topics. Its focus on poor-performing topics makes this a challenging task. This collection also has a remarkably large number of manual relevance judgments, containing $311k$ relevance judgments (over $1.2k$ per topic). This covers a large proportion of documents in the collection for each query, and makes the dataset less biased towards the original systems that participated in the competition because the judgment pooling depth was much larger. When training using this dataset, people often use 5-fold cross-validation, although there is no consensus in the community about which folds to use.

TREC WebTrack 2009–14 [24]. The TREC WebTrack focuses on retrieval from massive collections, as would be encountered when doing web search. There are 50 topics for each year, and two document collections explored (ClueWeb09 for years 2009–12, and ClueWeb12 for years 2013–14). This dataset is challenging due to its scale. It is common to perform round-robin evaluation with this dataset, testing on one year, validating on another, and training on the remaining years [76].

TREC CAR [47]. Inspired by the task of re-creating encyclopedic articles based on their heading structure, TREC Complex Answer Retrieval (CAR) frames this task as ad-hoc passage retrieval from Wikipedia. To encourage and facilitate neural system participation, the task includes 'automatic' relevance judgments assumed from the structure of Wikipedia articles: paragraphs under a particular heading (treated as a topic) are assumed to be relevant. This is a reasonable assumption due to the high editorial quality of Wikipedia. Since this only provides a notion of relevance, non-relevant documents must be assumed using some other technique. The track organizers also perform manual relevance assessment, which showed a correlation between performance of systems when evaluated on the automatic judgments and on the manual judgments.

MS-MARCO [132]. This dataset contains over $800k$ queries that were collected as questions found on the Bing query log. Annotators were instructed to write answers to questions based on passages retrieved from these documents. This dataset is used for several tasks, one of which is ad-hoc retrieval (passage re-ranking). Passages used to construct the answers are considered relevant, and all other documents are considered non-relevant. In this dataset, some topics have no relevant answers, which is thy the average number of relevance judgment per topic is less than 1. Due to the massive scale of this dataset, it is the base of the upcoming 2019 TREC Deep Learning

track. With the TREC task, a set of manual relevance judgments will be collected, enhancing the quality of the evaluation dataset.

A few trends can be observed from these datasets. First, the more queries the collection has, the fewer relevance judgments per topic tend to be available. This is a serious limitation for deep learning systems because non-scored documents are typically treated as non-relevant for most evaluations. This means that systems that identify novel relevant documents may be penalized for doing so. Since neural techniques address the problem of ad-hoc ranking in a new way, this potentially means that evaluations are biased against them. Furthermore, it could also bias the performance of these systems against finding novel relevant documents because such documents are not labeled as relevant. This is one reason why the Robust dataset is so valuable for neural ad-hoc ranking evaluation: it contains relevance judgments for a much higher proportion of documents in the collection per query than the other datasets. This, however, comes at the expense of a wide variety of topics and a large document collection. Because of this, weak supervision approaches for this task may be particularly valuable [43, 93]; models can be taught on an extensive and varied dataset (with few relevance judgments), but evaluated on a more robust dataset.

2.1.3 Neural Ranking Architectures

In this section, I present several proposed non-contextualized neural network architectures designed for the task of ad-hoc ranking.

Representation-Focused Matching

Representation-focused ranking architectures focus on modeling the representation function $\Phi(\cdot)$ using a deep neural network, and rely on a simple similarity function $F(\cdot)$. The representation function $\Phi(\cdot)$ produces two k-length vectors, one for the

query and one for the document: $\mathbf{r_q} = \Phi_q(\mathbf{q}) \in \mathbb{R}^k$, $\mathbf{r_d} = \Phi_d(\mathbf{d}) \in \mathbb{R}^k$. The representation function can be trained as part of a Siamese network (in this case $\Phi_q = \Phi_d$), or with a separate networks for learning the query and document independently. The similarity function is often just the cosine similarity between the two representations, i.e.,

$$F(\mathbf{r_q}, \mathbf{r_d}) = cosine(\mathbf{r_q}, \mathbf{r_d}) = \frac{\mathbf{r_q} \cdot \mathbf{r_d}}{|\mathbf{r_q}||\mathbf{r_d}|} = \frac{\sum_{i=1}^{k} r_{q_i} r_{d_i}}{\sqrt{\sum_{i=1}^{k} r_{q_i}^2} \sqrt{\sum_{i=1}^{k} r_{d_i}^2}} \tag{2.3}$$

I now present the most notable representation-focused ranking architectures.

DSSM [73]. The Deep Structured Semantic Model (DSSM), proposed in 2013, was the first representation-focused model for ad-hoc retrieval. It uses identical networks to produce $\mathbf{r_q}$ and $\mathbf{r_d}$. They start with a bag of words representation of each query/document in $\mathbb{N}^v$ where v is the size of the vocabulary and each value represents the count of occurrences of each word in the query/document (without normalization). Since this vector is very large and sparse, they reduce the dimensionality using a technique called word hashing. Each term is broken down into its component character trigrams (with one padding character on each side). Thus, they reduce their input space from $\mathbb{N}^v$ to $\mathbb{N}^t$, where t is the number of trigrams present in a vocabulary. For an English vocabulary of size $v = 500k$, this reduces the size to $t = 30k$. The authors show that collisions are rare empirically. Aside from reducing the dimensionality, this technique also accounts for some morphological differences in terms (e.g., 'cheese' and 'cheeses' contain common trigrams). These representations are further reduced into a lower-dimensional real-valued space using a 3-layer multi-layered preceptron. They propose intermediate layers of size 300, and a final layer of size 128. These representations are then compared using cosine similarity to produce relevance scores. This structure enables the model to learn the ways in which terms in a given query or document interact with one another, but does not account for how terms

may interact between the query and document (this is covered by interaction-focused models in Section 2.1.3). Due to the large number of parameters, this model needs a considerable amount of training data to perform well. Thus, the authors propose using query log clickthrough information to train this network using log likelihood loss. They show this is an effective approach, and outperform classical baselines like BM25 as well as other dimensionality-reduction techniques. However, these results are not directly replicable due to an evaluation conducted on a proprietary query log. Recently, DSSMs have also been applied to other natural language processing tasks.[2]

CDSSM [186]. This model builds upon the DSSM model by including convolution operations over terms in the document when building representations. Specifically, rather than treating entire queries/documents as a bag of character n-grams, it treats each word individually as a bag of character n-grams and performs a convolution operation over each window of k terms. The word n-gram scores are combined using max pooling, followed by a multi-layer perceptron to build the dense representations. As with DSSM, they recommend using cosine similarity to measure the distance between the query and document representations. The authors report ranking performance improvements compared to DSSM when this approach when using $k = 3$, and attribute the improvements to improved semantic understanding of local context. Their evaluation is conducted using a proprietary dataset based on a query log, so their results are not directly replicable. The paper also did not provide any analysis indicating that their model was successfully learning local contextual representations, rather than doing something else that is valuable but less interesting, e.g., learning

[2]For instance, Wang et al. [209] claim that DSSMs can be applied to commonsense reasoning. In this application, the authors replace the word hashing mechanism with word embeddings, the dense layers that build the representation with bi-directional recurrent layers (encoding different segments of a given sentence), and the scoring function with a multi-layer perceptron. Thus, their approach only loosely resembles the original DSSM (if at all).

to filter out stop words. They could have tested this by using smaller windows (e.g., $k = 1$, which would have shown if the model was learning to filter out individual terms), and by calculating the term similarity of example word n-grams that have similar meanings.

Interaction-Focused Matching

Interaction-focused ranking architectures focus on modeling the similarity function $F(\cdot)$ via a deep neural network, rather than the representation function $\Phi(\cdot)$. This is accomplished by using a simple approach to combine the query and document into interaction matrix $M \in \mathbb{R}^{|\mathbf{q}| \times |\mathbf{d}|}$ (sometimes called a similarity matrix or a translation matrix in literature), and learning how to transform this combined representation into the relevance score. Approaches primarily differ in the way in which they construct the interaction matrix, and the neural architectures that they use to transform the interaction matrix into a relevance score. The most common similarity matrix format is by using the cosine similarity between the word vector of each term in the query and each term in the document.

There have been a wide variety interaction-focused ranking architectures proposed. Here, I present the most notable efforts.

DRMM [56]. The Deep Relevance Matching Model made an important step for interaction-focused matching: it introduced the use of a cosine similarity matrix as the primary input source. Based on these scores, it proposed summarizing a query term's interaction with the document using histograms based on term similarity values. Buckets are placed between $[-1.0, 1.0]$ at intervals of 0.5, with one special bucket for exact matches with a similarity score of exactly 1.0. A simple densely-connected network combines this histogram for each query term into a relevance score for the query term. These term scores are then combined together to produce a final relevance

score via weighted summation. Two techniques were explored for term weighting, including by IDF and by term vector weighting (i.e., using a linear combination of the term word vectors). The author's experiments indicated that weighting by IDF performs better than term vector weighting. They also explored several normalization techniques for the histogram values itself. These techniques include normalizing by the document length (resulting in relative occurrence counts, rather than absolute counts), and by taking the logarithm of each count. They found that the length-normalized counts performed significantly worse than the other approaches, whereas the log count value performed marginally better than the raw count. They found that their approach outperforms the representation-focused approaches on the TREC Robust and WebTrack datasets. However, this is not a necessarily a fair comparison because it is well-known that representation-focused approaches require massive amounts of training data (such as query log behavioral information). They also showed that their approach significantly outperforms a Query Likelihood and BM25 baseline. In these experiments, BM25 and QL appeared to be adequately tuned, with performance comparable to those reported in [97].

KNRM [214]. The Kernel-based Neural Ranking Model is conceptually similar to DRMM in several ways. First, and most importantly, it replaces the hard buckets used by DRMM's histogram with soft buckets based on Gaussian kernels. These kernels function like the score histograms, but instead degrade their signals as the score is farther from the center. The kernel score k_i for query term i and document term j is:

$$k_i = \sum_j \exp\left(- \frac{(M_{i,j} - \mu)^2}{2\sigma^2} \right) \tag{2.4}$$

where μ and σ are parameters of the kernel. The authors use a bank of 10 kernels, evenly-spaced throughout the range of $[-0.9, 0.9]$ (with $\sigma = 0.1$), and a final narrow kernel around the identity ($\mu = 1.0$, $\sigma = 10^-3$). In their work, they used fixed kernels,

and apply back-propagation to the word vectors to shift them into the appropriate buckets. It was trained using proprietary query log information. In my own experiments on smaller public datasets (e.g., TREC Robust and WebTrack), I have found it more effective to keep the word vectors fixed and back-propagate to kernel parameters μ and σ. This results in the widening, narrowing, and shifting of the kernels to fit typical word similarity distributions. Like DRMM, KNRM use a linear combination of the scores to produce a relevance score for each query term. It then sums these individual scores to produce a final query relevance score. In their work, they do not use DRMM's approach of weighted score summation. Overall, this paper does an excellent job showing the effectiveness of the kernels, for instance by demonstrating how word similarity scores shift during training, and by showing which static values of μ are effective on their dataset.

ConvKNRM [37]. This model extends KNRM with convolutional filters over the input word embeddings. Through this mechanism, the model is able to learn n-gram similarity scores, an important consideration ignored by DRMM and KNRM. N-gram similarity scores are particularly important for multi-word expressions, such as 'New York City'; when each of these terms are treated independently, a document that describes a new construction project in the English city of York may be considered just as relevant as a document that mentions New York City itself. The model runs learned multi-channel 1-dimensional convolutions over the input word embeddings to produce n-gram embeddings. These embeddings are then 'cross-matched', meaning that a separate similarity matrix is generated for each n-gram length. For instance, if the maximum convolution size is 3, there will be 9 similarity matrices generated: one between unigrams and unigrams, one between unigrams and bigrams, etc. This allows some multi-word expressions to potentially match shorter variants. For instance, 'New York City' can match 'NYC'. The remainder of the pipeline is identical,

with the generalization of utilizing multiple similarity matrices. The authors demonstrate substantial gains on both WebTrack and a proprietary query log dataset, but unfortunately provide little in the way of analysis. Aside from a few case studies, they provide a strange evaluation in which scores from ConvKNRM are included as features in RankSVM to determine the relative importance. A much more effective and convincing way to demonstrate this would be to conduct an ablation study, which makes me skeptical that all the components are completely necessary.

MatchPyramid [146]. Like ConvKNRM, MatchPyramid also takes a convolutional approach to relevance ranking. However, unlike ConvKNRM, it captures patterns that occur in the similarity matrix itself by performing 2-dimensional convolution and pooling over the matrix. By stacking these convolutions on top of each other, the authors claim the model learns to capture larger and broader patterns throughout the document. Since the image gets smaller after each convolution layer, this forms a pyramid shape. This approach is borrowed from classic image processing techniques, which perform similar operations to detect edges, features, and objects in images. At the top of the pyramid, the scores are combined using a simple dense network into a final relevance score. The authors also experimented with alternative approaches for formulating the similarity matrix: using a dot product, and performing Gaussian filtering over the similarity scores. They found that Gaussian filtering was the most effective technique on the TREC Robust dataset, which severely limits noise from non-exact matches. They show that this does provide some benefits beyond only considering exact matches, however. The authors also experiment with various convolution and pooling sizes, finding that a large pooling size of 3×10 and a small convolution size of 1×3 is effective. These settings do not match my intuitions, particularly the convolution size; I'm not convinced that 1×3 convolutions would introduce any interesting patterns, besides just term occurrence; interesting patterns can occur

for adjacent terms. In related work, the authors also show that this technique is useful for other text matching tasks, such as paraphrase detection [147].

PACRR [75]. The Position-Aware Convolutional-Recurrent Relevance Matching model has some similarities to the MatchPyramid model, but aims to more explicitly model n-gram interactions between the query and document. It uses multiple sizes of learned square convolutional filters. These square filters are able to capture *soft n-grams*, referring to n-grams that may have soft term matches (via similarity scores) and/or are out-of-order or have skipped terms (e.g., a query of 'computer science course' could match 'courses in computer science'). The pooling strategy is different than MatchPyramid as well. Rather than pooling over document region, the model pools over the filter dimension, thus producing a single n-gram score for each query n-gram for each position in the document. To account for variable document lengths, the PACRR model then takes the top k scores along the document dimension for each query term (k-max pooling). In their experiments on TREC WebTrack, they showed that different values of k are valuable on different subsets, but reasonable values are $k = 2$ or $k = 3$. These query term n-gram scores are then concatenated with IDF scores, and aggregated using a recurrent neural network to produce a final relevance score. In follow-up work, the authors investigated the types of patterns captured by the model [224]. One such finding was that the model weights larger n-grams higher during score combination, presumably due to their rarity. One characteristic that makes this model unique is it's focus on top k matching (where k is a low number). This technique is at odds with conventional wisdom that the more frequently a term appears in a document, the more relevant it is. Nonetheless, the authors show encouraging results with their model. This is likely due to the enhanced notion of term proximity relevance that the model is able to capture.

CO-PACRR [76]. The CO-PACRR model is a variant of PACRR which attempts to better capture COntext. They accomplish this with three components: a query disambiguation component, cascaded k-max pooling, and a shuffled dense score combination step during training. The query disambiguation component measures the similarity score between each sliding window in the document and the query as a whole. This is accomplished by averaging the embeddings of the query, averaging the embeddings of the document window, and taking the cosine similarity between these values. These scores are concatenated with the n-gram scores before score combination. The authors claim that this allows the model to account for ambiguity in the term matches. Since this work was published, contextualized word vectors have gained popularity (e.g., BERT [44]), which may also be able to address this ambiguity. The cascaded k-max pooling operates by taking the k maximum values for varying proportions of terms in the document. The authors suggest using windows of 0-25%, 0-50%, 0-75%, and the full document. This cascading approach enables the model to identify whether strong term matches occur early in the document, which may indicate higher relevance than those that are only matched near the end of the document. Finally, they replace the recurrent combination of PACRR with a simple dense layer, as is done with other models like MatchPyramid. However, unlike other models, they shuffle the query term positions during training. This reduces the chance of overfitting to small training sets. The authors conduct extensive experiments on TREC WebTrack, and show that each component of their model is valuable via an ablation study. Overall, this paper contains many valuable insights that are overlooked by other neural ranking architectures. However, the authors could have strengthened the case for the modifications by also showing that they are effective on other models. For instance, perhaps cascaded pooling would also enhance KNRM, by summing the kernel scores over different document segments.

DeepRank [148]. The generically-named DeepRank model is conceptually similar to PACRR and CO-PACRR (yet it makes no reference to those works). However, it has some important distinctions. First, it limits the scope to only regions of the document that have term matches. This may reduce noise from elsewhere in the document. This is motivated by how humans may evaluate relevance of a document – they first jump to a particular keyword they are looking for, and then read the surrounding area of the document to determine if it's what they are looking for. Within each region, the authors experiment with both 2D (square) convolution and 2D GRU (recurrent) cells. They found that the convolution approach is more effective empirically. The scores are then combined using a recurrent layer that includes an encoding of document position. Similar to the cascaded pooling done by CO-PACRR, this can allow the model to prioritize matches that occur early in the document. The authors explore several ways to encode the position, and find an exponential or reciprocal encoding of document position is most effective (with tuned hyper-parameters). This is in line with prior work and intuition that the earlier that mentions occur in a document, the more important they are to a document's topic. They also found that including the word representations as additional input channels alongside the similarity matrix was more effective than the similarity matrix alone. This might be due to the use of a larger dataset (LETOR); the authors suggest that deep neural ranking models cannot be trained effectively on smaller datasets such as TREC Robust or WebTrack. However, as we know from the prior models covered in this section, neural rankers can be trained effectively on these datasets. Overall, although this paper shows promising results and provides some insights into what network characteristics can be valuable, the authors fail to place their contributions in the context of other related work done in the area.

DeepTileBars [193]. This work challenges the use of a similarity matrix as interaction input. Instead, the use a representation based on the TileBars algorithm [64], a technique for visualizing term occurrences in retrieved documents. The algorithm splits the document into segments based on topical shift. Thus, a document that goes on about one particular topic (e.g., spam) may be collapsed into a single segment, whereas a document that covers several topics or facets about a topic (e.g., a well-written encyclopedic article) may be represented in several topics. For each topic, the paper suggests three measurements for each query term: the term frequency of the topic in the given text segment, the IDF of the term (if it matches exactly anywhere in the segment), and the maximum cosine similarity of any term found in the segment. Then, 1-dimensional convolution is applied over the segments with varying lengths. The result of this convolution is combined using a recurrent neural network, and the final scores are combined using a simple dense layer. While I consider the intuition of the paper good, there are several ways they could have made their results more convincing. First, they did not demonstrate that their algorithm was tolerant to different TileBar parameters. An analysis showing the performance impact in the two degenerate situations for TileBars (one segment covering the entire document, and one segment for each document term) would also be valuable to see. Furthermore, the use of convolution places a heavy importance on query term order. Like CO-PACRR, the authors should have considered shuffling the query terms to avoid over-fitting to the training data, given that they evaluated on a dataset with relatively few queries (TREC WebTrack).

Hybrid Approaches

Not all matching architectures conform directly to the representation-interaction dichotomy. Here I cover notable examples which do not conform easily into the paradigm.

Duet [127]. The duet architecture combines an interaction model (which they call a 'local' model), and a representation model (which they call a 'distributed' model). Rather than combining existing approaches, they develop their own network topology. For the interaction model, they use a binary interaction matrix and perform convolution with a length of 1 over the query dimension, followed by dense score combination. Unlike other interaction models, this allows for learning exact absolute term occurrence positions in the document. Their representation model resembles CDSSM, but differs in the use of window-based pooling (window of 8 for Φ_q, and window of 100 for Φ_d).[3] This results in different sizes of query and document representations, where the query is represented as a 300-dimensional vector, and the document is represented as a 300×899 matrix, where 899 is derived from the maximum document length of the model. To compute the distance between the two, a Hadamard (element-wise) product is employed, followed by a multi-layer perceptron. The model produces a final relevance score simply by summing the scores of their two models. They train their system using behavioral information (Bing search engine click-through information), and evaluate on a subset of these queries that were manually scored for relevance. They show that their representation model is more effective on it's own than their

[3]The authors suggest that window-based pooling should perform better than global pooling because it allows for localized matching (e.g., maybe matches early in the document are more valuable than matches later in the document). Indeed, they show that the performance of just representation model slightly outperforms CDSSM. However, since other differences exist in the structure of the two (e.g., size of representations), it is unclear that the pooling alone has the effect.

interaction model (in fact, the interaction model barely outperforms a BM25 baseline), but combining the two models improves performance further.

Duet v2 [126]. The authors of the Duet model recently proposed a set of modifications to the duet model to tune the model for passage retrieval. As a technical report, the paper leaves much to be desired. However, through an ablation study on MS-MARCO, they provide evidence that the following changes to the Duet model can be effective for passage ranking:

1. Use an IDF-weighted binary interaction matrix. That is, rather than a simple binary indicator for each value, instead use the IDF of the term as the positive indicator. Among the changes, this improves the results most substantially.

2. Use multi-layer perceptron to combine score from representation and interaction models, rather than summing the scores.

3. Use `relu` activation function, rather than `tanh`. The authors provide no motivation for this change, but empirically show that it performs better.

4. Use pretrained word embeddings rather than character trigrams to build representation model.

5. Use bagging to produce an ensemble of Duet models, with different random seeds and a different sample of training data.

Weak Supervision and Domain Transfer

In IR, weak supervision uses pseudo-relevant information to train a ranking model in place of human judgments. This is valuable for overcoming the deep/wide dataset tradeoff and for building ranking models that are not overfit to a particular evaluation dataset.

Early work on weak supervision for IR focused on training learning-to-rank models [8], using web anchor text [6] and microblog hashtags [12] for weak supervision. More recently, Dehghani et al. [43] proposed a weak supervision approach that makes use of the AOL query log and BM25 results as a source of training data. Aside from limitations surrounding the availability of query logs, their approach suffers from limitations of BM25 itself: it assumes that documents ranked higher by BM25 are more relevant to the query than documents ranked lower. Others have suggested using a similar approach, but using news headlines [93], also assuming relevance from BM25 rankings. Still others have employed Generative Adversarial Networks (GANs) to build training samples [206], but this limits the generated data to the types of relevance found in the training samples, Others have demonstrated that one can transfer relevance signals across domains (e.g., from TREC Microblog to TREC Robust, as in [220]) making it a complementary approach. In contrast, in Section 2.2 I present an approach uses freely-available text *pairs* that exhibit both a high quality and large size.

2.2 Effective Ranking with Content-Based Weak Supervision

Our goal was to evaluate how well one can train neural ranking models on a large, naturally-occurring dataset of text pairs. The idea is that models successfully trained in such a way should be able to overcome a lack of adequate relevance training pairs, and also produce models that do not overfit a particular evaluation dataset. While others have explored weak supervision approaches for training ad-hoc ranking models, our work in [108, 112] uses text *pairs* as relevance signals (such as headlines and articles from news articles), rather than signals like BM25 (as done by [43]).

2.2.1 Methodology

Recall that pairwise training consists of a set of training triples, each consisting of a query q, relevant document d^+, and non-relevant document d^-. We describe two sources of weak supervision training data that replace human-generated relevance judgments: ranking-based and content-based training sources.

Ranking-based Sources

Ranking-based training sources, first proposed by [43], are defined by a collection of texts T, a collection of documents D, and an unsupervised ranking function $R(q, d)$ (e.g., BM25). Training triples are generated as follows. Each text is treated as a query $q \in T$. All documents in D are ranked using $R(\cdot)$, giving D^q. Relevant documents are sampled using a cutoff c^+, and non-relevant documents are sampled using cutoff c^-, such that $d^+ \in D^q[0 : c^+]$ and $d^- \in D^q[c^+ : c^-]$. This source is referred to as ranking-based because the unsupervised ranker is the source of relevance.[4]

Content-based Sources

Content-based training sources are defined as a collection of text pairs $P = \{(a_1, b_1), (a_2, b_2), ..., (a_{|P|}, b_{|P|})\}$ and an unsupervised ranking function $R(q, d)$ (e.g., BM25). The text pairs should be semantically related pairs of text, where the first element is similar to a query, and the second element is similar to a document in the target domain. For instance, they could be heading-content pairs of news articles (the headline describes the content of the article content). For a given text pair, a query and relevant document are selected $(q, d^+) \in P$. The non-relevant document

[4]Our formulation of ranking-based sources is slightly different than what was proposed by Dehghani et al. [43]: we use cutoff thresholds for positive and negative training samples, whereas they suggest using random pairs. Pilot studies we conducted showed that the threshold technique usually performs better.

is selected from the collection of documents in $B = \{b_1, b_2, ..., b_{|P|}\}$. We employ $R(\cdot)$ to select challenging negative samples from B^q. A negative cutoff c^- is employed, yielding negative document $d^- \in B^q[0 : c^-] - \{d^+\}$. We discard positive samples where d^+ is not within this range to eliminate overtly non-relevant documents. In a re-ranking setting, the neural ranker is unlikely to encounter documents so different from the query anyway. This approach can yield documents relevant to q, but we assert that d^+ is *more* relevant.

Although ranking-based and content-based training sources bear some similarities, important differences remain. Content-based sources use text pairs as a source of positive relevance, whereas ranking-based sources use the unsupervised ranking. Furthermore, content-based sources use documents from the pair's domain, not the target domain. We hypothesize that the enhanced notion of relevance that content-based sources gain from text pairs will improve ranking performance across domains, and show this in Section 2.2.2.

Filter Framework

We propose a filtering framework to overcome domain mismatch that can exist between data found in a weak supervision training source and data found in the target dataset. The framework consists of a filter function $F_D(q, d)$ that determines the suitability of a given weak supervision query-document pair (q, d) to the domain D. All relevant training pairs $(q, d^+) \in S$ for a weak supervision source S are ranked using $F_D(q, d^+)$ and the c_{max} maximum pairs are chosen: $S_D = \max^{c_{max}}_{(q,d^+)\in S} F_D(q, d^+)$. To tune $F_D(\cdot)$ to domain D, a set of *template pairs* from the target domain are employed. The set of pairs T_D is assumed to be relevant in the given domain.[5] We assert that

[5]Templates do not require human judgments. We use sample queries and an unsupervised ranker to generate T_D. Manual judgments can be used when available.

these filters are easy to design and can have broad coverage of ranking architectures. We present two implementations of the filter framework: the $k$max filter, and the Discriminator filter.

k-Maximum Similarity ($k$max) filter. This heuristic-based filter consists of two components: a *representation function* $rep(q, d)$ and a *distance function* $dist(r_1, r_2)$. The representation function captures some matching signal between query q and document d as a vector. Since many neural ranking models consider similarity scores between terms in the query and document to perform soft term matching [37, 56, 75, 214], this filter selects the k maximum cosine similarity scores between the word vectors of each query term and all terms in the document: $\max^{k}_{d_j \in d} sim(q_i, d_j) : \forall q_i \in q$.

Since neural models can capture local patterns (e.g., n-grams), we use an aligned mean square error. The aligned MSE iterates over possible configurations of elements in the representation by shifting the position to find the alignment that yields the smallest distance. In other words, it represents the minimum mean squared error given all rotated configurations of the query. Based on the shift operation and given two interaction representation matrices r_1 and r_2, the aligned $dist_{kmax}(r_1, r_2)$ is defined as the minimum distance when shifting r_1 for $s \in [1, |r_1|)$. More formally: $dist_{kmax}(r_1, r_2) = \min_{s=1}^{|r_1|} MSE\big(shift(r_1, s), r_2\big)$.

Using these two functions, the filter is simply defined as the minimum distance between the representations of it and any template pair from the target domain:

$$F_D(q, d) = \min_{(q', d') \in T_D} dist(rep(q, d), rep(q', d')) \tag{2.5}$$

Discriminator filter. A second approach to interaction filtering is to use the ranking architecture R itself. Rather than training R to distinguish different degrees of relevance, here we use R to train a model to distinguish between samples found

in the weak supervision source and T_D. This technique employs the same pairwise loss approach used for relevance training and is akin to the discriminator found in generative adversarial networks. Pairs are sampled uniformly from both templates and the weak supervision source. Once R_D is trained, all weak supervision training samples are ranked with this model acting as $F_D(\cdot) = R_D(\cdot)$.

The intuition behind this approach is that the model should learn characteristics that distinguish in-domain pairs from out-of-domain pairs, but it will have difficulty distinguishing between cases where the two are similar. One advantage of this approach is that it allows for training an interaction filter for any arbitrary ranking architecture, although it requires a sufficiently large T_D to avoid overfitting.

2.2.2 Experiment

We now evaluate the effectiveness of models trained with ranking-based, content-based, and filtered weak supervision.

Experimental Setup

Training sources. We use the following four sources of training data to verify the effectiveness of our methods:

- **Query Log (AOL, ranking-based, $100k$ queries).** This source uses the AOL query log [149] as the basis for a ranking-based source, following the approach of [43].[6] We retrieve ClueWeb09 documents for each query using the Indri[7] query

[6]Distinct non-navigational queries from the AOL query log from March 1, 2006 to May 31, 2006 are selected. We randomly sample $100k$ of queries with length of at least 4. While Dehghani et al. [43] used a larger number of queries to train their model, the state-of-the-art relevance matching models we evaluate do not learn term embeddings (as [43] does) and thus converge with fewer than $100k$ training samples.

[7]https://www.lemurproject.org/indri/

likelihood (QL) model. We fix $c^+ = 1$ and $c^- = 10$ due to the expense of sampling documents from ClueWeb.

- **Newswire (NYT, content-based, $1.8m$ pairs).** We use the New York Times corpus [176] as a content-based source, using headlines as pseudo queries and the corresponding content as pseudo relevant documents. We use BM25 to select the negative articles, retaining top $c^- = 100$ articles for individual headlines.

- **Wikipedia (Wiki, content-based, $1.1m$ pairs).** Wikipedia article heading hierarchies and their corresponding paragraphs have been employed as a training set for the TREC Complex Answer Retrieval (CAR) task [129]. We use these pairs as a content-based source, assuming that the hierarchy of headings is a relevant query for the paragraphs under the given heading. Heading-paragraph pairs from train fold 1 of the TREC CAR dataset [47] (v1.5) are used. We generate negative heading-paragraph pairs for each heading using BM25 ($c^- = 100$).

- **Manual relevance judgments (WT10).** We compare the ranking-based and content-based sources with a data source that consists of relevance judgments generated by human assessors. In particular, manual judgments from 2010 TREC Web Track ad-hoc task (WT10) are employed, which includes $25k$ manual relevance judgments ($5.2k$ relevant) for 50 queries (topics + descriptions, in line with [56, 75]). This setting represents a new target domain, with limited (yet still substantial) manually-labeled data.

Training neural IR models. We test our method using several state-of-the-art neural IR models: PACRR [75], Conv-KNRM [37], and KNRM [214].[8] We use

[8]By using these state-of-the-art architectures, we are using stronger baselines than those used in [43, 93].

the model architectures and hyper-parameters (e.g., kernel sizes) from the best-performing configurations presented in the original papers for all models. All models are trained using pairwise loss for 200 iterations with 512 training samples each iteration. We use Web Track 2011 (WT11) manual relevance judgments as validation data to select the best iteration via nDCG@20. This acts as a way of fine-tuning the model to the particular domain, and is the only place that manual relevance judgments are used during the weak supervision training process. At test time, we re-rank the top 100 Indri QL results for each query.

Interaction filters. We use the 2-maximum and discriminator filters for each ranking architecture to evaluate the effectiveness of the interaction filters. We use queries from the target domain (TREC Web Track 2009–14) to generate the template pair set for the target domain T_D. To generate pairs for T_D, the top 20 results from query likelihood (QL) for individual queries on ClueWeb09[9] and ClueWeb12,[10] are used to construct query-document pairs. Note that this approach makes no use of manual relevance judgments because only query-document pairs from the QL search results are used (without regard for relevance). We do not use query-document pairs from the target year to avoid any latent query signals from the test set. The supervised discriminator filter is validated using a held-out set of 1000 pairs. To prevent overfitting the training data, we reduce the convolutional filter sizes of PACRR and ConvKNRM to 4 and 32, respectively. We tune c_{max} with the validation dataset (WT11) for each model ($100k$ to $900k$, $100k$ intervals).

Baselines and benchmarks. As baselines, we use the AOL ranking-based source as a weakly supervised baseline [43], WT10 as a manual relevance judgment baseline, and BM25 as an unsupervised baseline. The two supervised baselines are trained

[9] `https://lemurproject.org/clueweb09.php`
[10] `https://lemurproject.org/clueweb12.php`

using the same conditions as our approach, and the BM25 baselines is tuned on each testing set with Anserini [217], representing the best-case performance of BM25.[11] We measure the performance of the models using the TREC Web Track 2012–2014 (WT12–14) queries (topics + descriptions) and manual relevance judgments. These cover two target collections: ClueWeb09 and ClueWeb12. Akin to [43], the trained models are used to re-rank the top 100 results from a query-likelihood model (QL, Indri [190] version). Following the TREC Web Track, we use nDCG@20 and ERR@20 for evaluation.

RESULTS

In Table 2.2, we present the performance of the rankers when trained using content-based sources without filtering. In terms of absolute score, we observe that the two n-gram models (PACRR and ConvKNRM) always perform better when trained on content-based sources than when trained on the limited sample of in-domain data. When trained on NYT, PACRR performs significantly better. KNRM performs worse when trained using the content-based sources, sometimes significantly. These results suggest that these content-based training sources contain relevance signals where n-grams are useful, and it is valuable for these models to see a wide variety of n-gram relevance signals when training. The n-gram models also often perform significantly better than the ranking-based AOL query log baseline. This makes sense because BM25's rankings do not consider term position, and thus cannot capture this important indicator of relevance. This provides further evidence that content-based sources do a better job providing samples that include various notions of relevance than ranking-based sources.

[11] Grid search: $b \in [0.05, 1]$ (0.05 interval), and $k_1 \in [0.2, 4]$ (0.2 interval)

Table 2.2 Ranking performance when trained using content-based sources (NYT and Wiki).

		nDCG@20		
Model	Training	WT12	WT13	WT14
BM25 (tuned w/ [217])		0.1087	0.2176	0.2646
PACRR	WT10	B↑ 0.1628	0.2513	0.2676
	AOL	0.1910	0.2608	0.2802
	NYT	**W↑ B↑ 0.2135**	**A↑ W↑ B↑ 0.2919**	**W↑ 0.3016**
	Wiki	W↑ B↑ 0.1955	A↑ B↑ 0.2881	W↑ 0.3002
Conv-KNRM	WT10	B↑ 0.1580	0.2398	B↑ 0.3197
	AOL	0.1498	0.2155	0.2889
	NYT	**A↑ B↑ 0.1792**	**A↑ W↑ B↑ 0.2904**	**B↑ 0.3215**
	Wiki	0.1536	A↑ 0.2680	B↑ 0.3206
KNRM	WT10	B↑ 0.1764	**0.2671**	0.2961
	AOL	**B↑ 0.1782**	0.2648	**0.2998**
	NYT	W↓ 0.1455	A↓ 0.2340	0.2865
	Wiki	A↓ W↓ 0.1417	0.2409	0.2959

Significant differences compared to the baselines ([B]M25, [W]T10, [A]OL) are indicated with ↑ and ↓ (paired t-test, $p < 0.05$).

When comparing the performance of the content-based training sources, we observe that the NYT source usually performs better than Wiki. We suspect that this is due to the web domain being more similar to the newswire domain than the complex answer retrieval domain. For instance, the document lengths of news articles are more similar to web documents, and precise term matches are less common in the complex answer retrieval domain.

We present filtering performance on NYT and Wiki for each ranking architecture in Table 2.3. In terms of absolute score, the filters almost always improve the content-based data sources, and in many cases this difference is statistically significant. The one exception is for Conv-KNRM on NYT. One possible explanation is that the filters caused the training data to become too homogeneous, reducing the ranker's ability to generalize. We suspect that Conv-KNRM is particularly susceptible to this problem because of language-dependent convolutional filters; the other two models rely only on term similarity scores. We note that Wiki tends to do better with the 2max filter, with significant improvements seen for Conv-KNRM and KNRM. In thse models, the discriminator filter may be learning surface characteristics of the dataset, rather than more valuable notions of relevance. We also note that c_{max} is an important (yet easy) hyper-parameter to tune, as the optimal value varies considerably between systems and datasets.

2.2.3 Summary

We presented an approach for employing content-based sources of pseudo relevance for training neural IR models. We demonstrated that our approach can match (and even outperform) neural ranking models trained on manual relevance judgments and existing ranking-based weak supervision approaches using two different sources of data. We also showed that performance can be boosted using two filtering techniques:

Table 2.3 Ranking performance using filtered NYT and Wiki.

			WebTrack 2012–14	
Model	Training	k_{max}	nDCG@20	ERR@20
PACRR	NYT		0.2690	0.2136
	w/ 2max	$200k$	0.2716	0.2195
	w/ discriminator	$500k$	↑ **0.2875**	**0.2273**
	Wiki		0.2613	0.2038
	w/ 2max	$700k$	0.2568	0.2074
	w/ discriminator	$800k$	**0.2680**	**0.2151**
Conv-KNRM	NYT		0.2637	0.2031
	w/ 2max	$100k$	↓ 0.2338	**0.2153**
	w/ discriminator	$800k$	**0.2697**	0.1937
	Wiki		0.2474	0.1614
	w/ 2max	$400k$	**0.2609**	↑ **0.1828**
	w/ discriminator	$700k$	0.2572	0.1753
KNRM	NYT		0.2220	0.1536
	w/ 2max	$100k$	0.2235	↑ **0.1828**
	w/ discriminator	$300k$	**0.2274**	↑ 0.1671
	Wiki		0.2262	0.1635
	w/ 2max	$600k$	↑ **0.2389**	↑ **0.1916**
	w/ discriminator	$700k$	0.2366	0.1740

Significant improvements and reductions compared to unfiltered dataset are marked with ↑ and ↓ (paired t-test, $p < 0.05$).

one heuristic-based and one that re-purposes a neural ranker. By using our approach, one can effectively train neural ranking models on new domains without behavioral data and with only limited in-domain data. This demonstrates that one can use alternative data sources to overcome poor training data.

2.3 Employing Dataset Characteristics

It is common to use search technologies to find answers to questions. While considerable work has investigated techniques to answer questions that have factoid answers, there has been less focus on open-ended questions that cannot be answered with simple, standalone facts. The information need of these open-ended questions are fulfilled with *complex answers*, which contain comprehensive details and context pertaining to the question. Thus, *Complex Answer Retrieval* (CAR) is the process of finding answers to questions that have complex answers [47].[12] More formally, given a question and a large collection of candidate answers, a CAR engine retrieves and ranks the answers by relevance to the question. Since candidate answers include details and context, they can be formulated as paragraphs of text. In this work, we propose and evaluate approaches for improving CAR by using structural and frequency information from the query and information from a knowledge graph constructed from CAR training data.

CAR queries can be broken down into two components: the topic and facet. The topic is the main entity of the question. For the query *'Is cheese healthy?'*, the topic

[12]Note that CAR queries are not necessarily complex. A question as simple as *'Is cheese healthy?'* requires a complex answer: a detailed and nuanced description of positive and negative health effects of cheese consumption is required to satisfy the information need. In contrast, a question such as *'How much Mozzarella cheese do I need to eat to satisfy my daily requirement of calcium?'* is a complex question with a simple factoid answer because it involves advanced reasoning that goes beyond what is typically captured by a knowledge graph.

is the entity 'Cheese' (see Figure 2.1). All answers to the question must be about this entity, otherwise the answer is not valid. The facet is the particular detail about which the question inquires. Question facets differentiate CAR queries from topical queries by inquiring about a specific detail about the topic. In the example, the facet can be described as 'Health effects'. For an answer to be considered valid to the question, it must refer to health effects of cheese—information such as nutrition, health risks, etc. These answers can come from multiple sources. For instance, an article about cardiovascular disease may claim that diets containing foods such as cheese that are high in saturated fat increase one's risk of heart disease (Answer 2 in Figure 2.1). Such a paragraph would be valuable to include in a complete answer because it

---

```
Question: Is cheese healthy?
Topic: Cheese
Facet: Health effects

A1: Although nutrient levels vary by type of cheese, most cheese are a
    rich source of calcium, protein, and sodium. Because the primary
    ingredient of cheese is milk, most of most of the nutritional...
A2: Consumption of foods that are high in saturated fat such as cheese
    are linked with an increase risk of cardiovascular disease.
    This is due to adverse effects to blood lipids and circulating...
A3: Cheese has the potential for promoting the growth of Listeria
    bacteria. This can cause serious infection in an infant and
    woman and can be transmitted to her infant in utero or after...
A4: Wisconsin is known as "America's Dairyland" because it is one of
    the nation's leading dairy producers, particularly famous for its
    cheese. Manufacturing, health care, information technology, and...
```

Figure 2.1 Hypothetical CAR query. Includes three relevant answers (1-3) and one non-relevant answer (4). Term matches from the topic and facet are underlined.

contains contextual information about why cheese consumption can increase one's risk of heart disease. Thus, it follows to frame CAR as retrieval of paragraphs of text from authoritative sources given a topic and facet. CAR queries with various facets about a single topic can be combined to produce a detailed article about a topic.

A straightforward approach for CAR is to use an existing IR technique as-is, concatenating the topic and facet information to build the query. Indeed, previous results showed that this basic approach can be effective [129], particularly when neural models are employed. However, such an approach is limited by several factors specific to CAR. We observe that facets are not necessarily mentioned verbatim in relevant paragraphs. In Figure 2.1, the relevant answers (1-3) never include 'health' because the context is clear from the other entities mentioned in the answer. On the other hand, non-relevant answers sometimes do include the facet term (e.g., answer 4). An effective CAR engine needs to account for this. Not all facets exhibit this general behavior. Let's consider another query: *"What is the effect of curdling in cheese production?"* (topic: cheese, facet: curdling). Since 'curdling' is not a general-language term, relevant answers probably need to use the term itself; related entity mentions are probably inadequate and would result in confusing text. We refer to this distinction as *facet utility*: high-utility facets use language that is specific to the topic and can be found directly in relevant answers (e.g., 'curdling'), whereas low-utility facets use general language and requires additional domain knowledge to identify relevant paragraphs (e.g., 'health effects').

Given these observations, we propose a two-pronged approach to CAR. First, we attempt to predict facet utility. For this, we use both structural information about the query itself, and corpus statistics about how frequently facets are used. Then, to better accommodate low-utility facets, we utilize entity mentions in the candidate answer. To this end, we construct a knowledge graph embeddings that contain facet

context using a training corpus, and measure distances between the topic entity and the entity mentions. We incorporate both the facet utility estimators and the entity scores into a neural ranking model, and use the model to retrieve complex answers. These approaches yield significant improvements over the unmodified neural ranker, and up to 26% improvement over the next best approach. We then provide a detailed analysis of our results, which shows that low-utility facets are indeed more difficult to match, and that our approach improves these results. In summary, our contributions are as follows:

2.3.1 Background

Complex answer retrieval is a new area of research in IR. In 2017, the TREC conference ran a new shared task focused on CAR [47]. The goal of this task is to rank answer paragraphs corresponding to a complex question. The shared task frames CAR in terms of Wikipedia content generation. This is appropriate because the editorial guidelines and strong role of moderators makes Wikipedia an authoritative source of information [65]. Paragraphs from articles meeting topic criteria[13] are selected as a source of answers for retrieval. The task goes one step farther by asserting that Wikipedia is also a good source of CAR queries and relevance judgments. CAR queries are formed from article titles (the query topic, an entity), and headings (query facets of that particular topic). Figure 2.2 shows an article's heading hierarchy, including the question topic and an example facet. Furthermore, paragraphs found under each heading are treated as automatically relevant to that particular topic, yielding a large amount of training data [45]. This makes it practical to train data-hungry neural IR models for CAR.

[13]E.g., templates, talk pages, portals, lists, references, and pages representing people, organizations, music, books, and others are discarded [47].

Table 2.4 Example CAR queries from Wikipedia by heading position.

Title (Q_1)		Intermediate Heading(s)		Target Heading (Q_n)	n
Cheese	»	*(none)*	»	Nutrition and health	2
Green sea turtle	»	Ecology and behavior	»	Life cycle	3
History of the United States	»	20th Century	»	Imperialism	3
Disturbance (ecology)	»	*(none)*	»	Cyclic disturbance	2
Medical tourism	»	Destinations » Europe	»	Finland	4

Queries vary in the number of headings included, because some queries have a different number of intermediate headings.

TREC CAR exploits the hierarchical nature of headings by using multiple headings to form each query. Let Q be a CAR query, consisting of n headings $\{Q_1, Q_2, ..., Q_n\}$. Let Q_n be the *target heading* of the query (that is, the primary facet of the query), Q_1 be the *title* of the article containing the heading (representing the topic entity of the query), and $\{Q_2, Q_3, ..., Q_{n-1}\}$ be any *intermediate headings* along the shortest path between the title and target heading in the heading hierarchy. Intermediate headings are important to provide adequate context for target heading. For instance, in Figure 2.2, heading 7.3 refers specially to the pasteurization of cheese as it relates to nutrition and health. Table 2.4 provides example queries using this terminology. Note that a given query does not necessarily have any intermediate headings, and that a target heading in one query (*Cheese » Nutrition and health*)[14] can appear as an intermediate heading in another query (*Cheese » Nutrition and health » Pasteurization*) because all headings containing paragraphs directly are treated as the target heading of a query.

[14]We use the symbol » to separate heading components of a query.

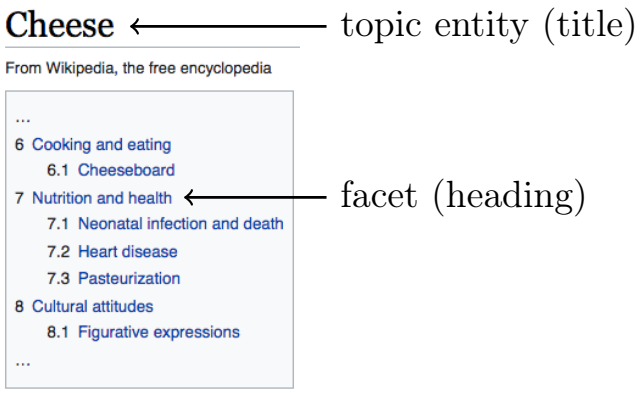

Figure 2.2 Sample headings found in a Wikipedia article. The article is titled *Cheese*, and it includes labels for the question "*Is cheese healthy?*".

Existing work in CAR is relatively limited. Prior to TREC, Nanni et al. [129] investigated baseline approaches using the automatic relevance judgments (assuming only paragraphs found under a heading are relevant to that query). Their approaches include BM25, cosine similarity (both with TF-IDF vectors and word embeddings), a baseline learning-to-rank approach, query expansion, and a deep neural model. They found that the deep neural model Duet [127] outperforms the others.

The 2017 TREC CAR task inspired several new approaches to CAR. A top-performing submission uses a Sequential Dependence Model (SDM [125]) for answer retrieval [100]. They modified the SDM for CAR by considering ordered n-grams that occur within an individual heading component, and unordered n-grams to terms in different headings. This results in a bias toward matching partial phrases that appear in individual headings, and reduces processing time for long queries. Another approach uses a Siamese attention network [122]. Features incorporated into this network include abbreviated entity names and lead paragraph entity mentions from

DBPedia [7]. Yet another approach uses reinforcement learning query reformulation for CAR [134, 137]. The query reformulation step helps the model add terms that make up for low-utility facets. Our approach differs from these by (1) explicitly modeling facet utility to predict which headings are unlikely to appear in relevant documents, and (2) incorporating contextualized entity relatedness measures to improve performance on low-utility facets.

Wikipedia and Knowledge Graphs

Since knowledge graphs encode which entities are related to one another, they may be valuable when identifying paragraphs with low-utility facets. There have been many efforts to organize Wikipedia information into a knowledge graph. Prominent efforts have been the DBPedia [7] and Freebase [13]. Researchers have found that query expansion using knowledge bases can improve ad-hoc retrieval [40, 212]. Others have investigated how to include knowledge graph features directly in learning-to-rank approaches [180]. More recently, Xiong et al. [213] explored an approach for including entity information in interaction-focused neural rankers by including additional entity-term and entity-entity similarity scores during matching, and an attention mechanism to deal with entity uncertainty. In contrast, in this work we exploit the topic and facet context information provided by CAR queries when modeling entity-entity similarity, and propose an approach for generating a knowledge graph suitable for training these embeddings.

Other efforts have been in how to train embeddings for entities and relations in knowledge graphs. Prominent approaches include translational embeddings (TransE) [14], translational hyperplane embeddings (TransH) [210], and holographic embeddings (HolE) [133]. Knowledge graphs have also been employed extensively for general question answering tasks [187, 226]. In this work, we observe that entity

similarity act as a signal for answers that are otherwise difficult to match (i.e., when facets are low-utility). We build a knowledge graph from Wikipedia articles using entity mentions and heading labels. We then train embeddings, and use embedding similarities when ranking answers.

2.3.2 Methodology

As mentioned in Section 2.3.1, complex answer retrieval (CAR) is a new IR task focused on retrieving complex answers to questions that include a topic and facet. Paragraphs from authoritative resources (e.g., Wikipedia) are considered partial answers for these questions. Identifying and overcoming low-utility facets is a central challenge to CAR. Here, we propose approaches based on the Wikipedia-focused CAR problem:

- *High-utility facets.* We find that high-utility facets correspond to headings that relate to specific details of an article's topic: *topical headings*. Thus, a given topical heading is unlikely to appear in most articles. However, we predict that terms found in topical headings are *more* likely to appear in relevant paragraphs because people are less likely to have the existing knowledge to determine meaning from the mentioned entities. Since the title of an article is the article's topic, it is necessarily topical.

- *Low-utility facets.* We identify that low-utility facets often correspond to structural headings in Wikipedia. These headings provide coherence across articles by enforcing a predictable document structure. As a result, these headings occur frequently. Furthermore, they often appear as intermediate headings by organizing topical headings below them. Since the terminology is necessarily more general, we predict that terms in structural headings are less likely to appear

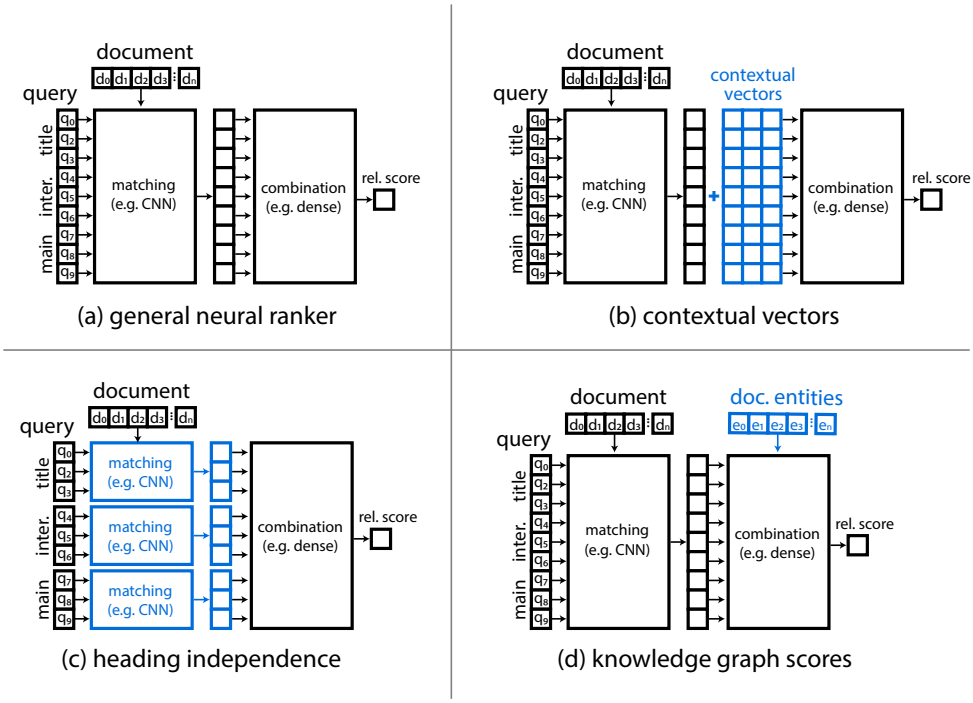

Figure 2.3 Example ranking architecture adaptations for CAR. (a) General ranking architecture, with matching and combination phases (unmodified). (b) Modified architecture, including contextual vectors for combination. (c) Modified architecture, splitting for heading independence. (d) Modified architecture, including knowledge graph scores in the combination layer.

in relevant paragraphs because readers can figure out the context by related entities. Thus, we propose including additional entity similarity information to accommodate these cases.

Recall that an interaction-focused neural ranker (Figure 2.3a) involves two phases: a matching phase that identifies places in the document in which query terms are used (e.g., convolution), and a combination phase in which the scores for each query term are combined to produce a final relevance score (e.g., dense). In the remainder of this section, we present two approaches to inform an arbitrary interaction-focused neural

Table 2.5 Sample contextual vectors for CAR.

	green	sea	turtle	ecology	and	behavior	life	cycle
position_title	1	1	1	0	0	0	0	0
position_intermediate	0	0	0	1	1	1	0	0
position_target	0	0	0	0	0	0	1	1
heading_frequency	0	0	0	3	3	3	3	3

Example is for the query "*green sea turtle » ecology and behavior » life cycle*".

ranker of facet utility (Section 2.3.2-2.3.2, Figure 2.3b-c), and one approach to include entity similarity signals to inform the model of relevance for low-utility facets using knowledge base embedding similarity scores (Section 2.3.2, Figure 2.3d). Finally, we describe our implementation strategy for two concrete neural rankers (PACRR and K-NRM, Section 2.3.2).

Contextual Vectors

Many information retrieval approaches use the simple (yet powerful) term IDF value as a signal for query term importance. Neural models incorporate this signal, too. For instance, DRMM [56] and PACRR [75] use an IDF vector in the combination phase. We generalize this approach by allowing an arbitrary number of contextual vectors to be included in the model alongside term interaction signals (Figure 2.3b). Here, we use contextual vectors to provide an estimation of facet utility, understanding that the models can learn how to use these values when making relevance judgments. We propose two such estimators: *heading position* (HP) and *heading frequency* (HF).

Heading position Recall that CAR queries consist of a title, intermediate headings, and a target heading. When estimating heading utility, the position of the heading in the list intuitively provides a signal of heading utility. For instance, the title is necessarily topical; it is the topic itself. Furthermore, any intermediate headings are likely structural because they provide categorizations for the target heading. Finally, the target heading may be either topical or structural; its position inherently tells the model nothing about whether or not to expect the term to appear. Thus, we encode three vectors to encode query term position information: one that indicates if the term exists in the title, and intermediate heading, or the target heading. An example of these vectors is given in Table 2.5. We also considered using query depth to capture the hierarchical information (i.e., title, level 1 heading, level 2 heading, etc.). However, the drawback of using depth information is that it is variable by article. For example, while some articles use level 1 headings as purely structural components, others use content-based level 1 headings. By using the heading position approach, all structural intermediate headings are grouped together, allowing the model to more easily learn to always treat them as context for the target heading.

The benefits of including heading position information can extend beyond simply distinguishing whether a term is likely in a topical or structural heading. For instance, the topic of a question may be abbreviated in relevant paragraphs to avoid excessive repetition. By including heading position information in this way, the model can distinguish when certain matching patterns are important to capture.

Heading frequency Another possible estimator of heading utility is the frequency that a heading appears in a sufficiently representative dataset. For instance, because structural headings such as 'Nutrition and health' are general and can accommodate a variety of topics, one would expect to find the heading in many food-related articles.

Indeed, the heading appears in Wikipedia articles such as *Cheese*, *Beef*, *Raisin*, and others. On the other hand, one would expect topical headings to use less general language, and thus be less likely to appear in other articles. For instance the heading 'After the Acts of Union of 1707' only appears in the article *United Kingdom*.

To represent this value, we use the document frequency of each heading: the *heading frequency*. Thus, frequent headings (e.g., 'History') have a large heading frequency value, and infrequent headings (e.g., 'After the Acts of Union of 1707') have a low value. For heading matching, we require a complete, case-insensitive match of the text as a heading in an article. Thus, 'Health effects' and 'Health Effects' are considered the same heading (capitalization), but not 'Health effects' and 'Health' (substring), 'Health effect' (difference in pluralization), or 'Health affects' (typographical error). To group similarly-frequent headings together, we stratify the heading frequency value when used as a contextual vector in a range of $[0, 3]$, which allows easy incorporation into neural models. Based on tuning conducted in pilot studies, we found effective breakpoints to be the 60th, 90th, and 99th percentiles.[15] An example of the heading frequency vector is given in Table 2.5.

When the model encounters the heading frequency value, we expect it to learn to value high-frequency headings less than low-frequency headings (i.e., assign a negative weight). This results in similar behavior to IDF, but with the important distinction that it operates over entire headings, rather than terms. For instance, most terms in 'After the Acts of Union of 1707' have a low IDF (i.e., they appear frequently), but as a whole heading, it appears infrequently and is likely important to match.

[15]For reference, the 60th percentile is approximately the cutoff for headings that only appear a few times such as *Red Hot Chili Peppers*; the 90th percentile is approximately the cutoff of moderately frequent headings such as *Finland*; and the 99th percentile is approximately the cutoff of frequent headings such as *Family and personal life*.

Combining contextual vectors The two contextual vectors capture different notions of heading utility. Heading position contextual vectors are able to discriminate otherwise identical headings based on where they appear in the query. This can be valuable in some situations: for instance, 'Imperialism' and 'Finland' in Table 2.4 appear as target headings, but they could also appear as the topic of other queries. Heading frequency contextual vectors treat headings with the same text identically, regardless of their position in the query. However, their power comes from informing the model about the nature of the specific heading, and can help identify the utility of target headings (which are otherwise ambiguous), or headings that do not match the typical characteristics in other heading positions. By including both vectors, the model should be able to learn a better sense of heading utility by combining the two notions.

Heading Independence

Although contextual vectors can help inform a trained ranker about which headings are structural and topical, they cannot directly affect which types of interactions are identified and how these interactions are scored because this occurs earlier in the pipeline of the neural model considered (matching phase). We hypothesize that heading utility can actually affect which signals are important. For instance, a low-utility structural heading might be more tolerant to weak term similarities (i.e., 'colonial' or 'ancient' might be acceptable matches for 'History'), or a high-utility topical heading might have stricter requirements for maintaining the order of terms. Thus, we propose a general approach to modify a neural ranking architecture to adjust matching based on heading utility: *heading independence*.

Heading independence involves splitting the matching phase of a neural ranker into multiple segments, each responsible for processing a segment of the query. Here, we split the query by heading position: title, intermediate, and target headings (see

Figure 2.3c). The processing of each heading position is independent of the others, so a different set of parameters can be learned for each component. The results of the matching layers are then concatenated and sent to the combination layer of the unmodified model (e.g., a dense layer) for the calculation of the final relevance score.

This approach makes intuitive sense because one would expect the title to interact differently in the document than a structural or topical heading. For instance, abbreviated versions of the topic often are used to improve readability. Furthermore, this approach enforces alignment of headings during combination (see Figure 2.4). When all query terms are simply concatenated, the alignment of each query position will change among queries due to differences in query length. Thus, the model will need to learn and accommodate multiple tendencies in a wider variety of positions of the query, rather in just certain areas.

Knowledge Graph

Although interaction-focused neural rankers often employ word embedding similarities, word embeddings do not always capture similarities within contexts. This results

```
Q1: Cheese (Q1) > Nutrition (Qn) and (Qn) health (Qn)
Q2: Medical (Q1) tourism (Q1) > Destinations (Q2) > Europe (Q2) > Finland (Qn)

              Unaligned                      Aligned
    --------------------------     --------------------------
Q1: Q1 Qn Qn Qn __ __ __ __ __     Q1 __ __ __ __ __ Qn Qn Qn
Q2: Q1 Q1 Q2 Q2 Qn __ __ __ __     Q1 Q1 __ Q2 Q2 __ Qn __ __
```

Figure 2.4 Example term alignment benefits of using heading independence with two sample queries. Q1: query title. Q2: intermediate heading. Qn: target heading. __ indicates padding.

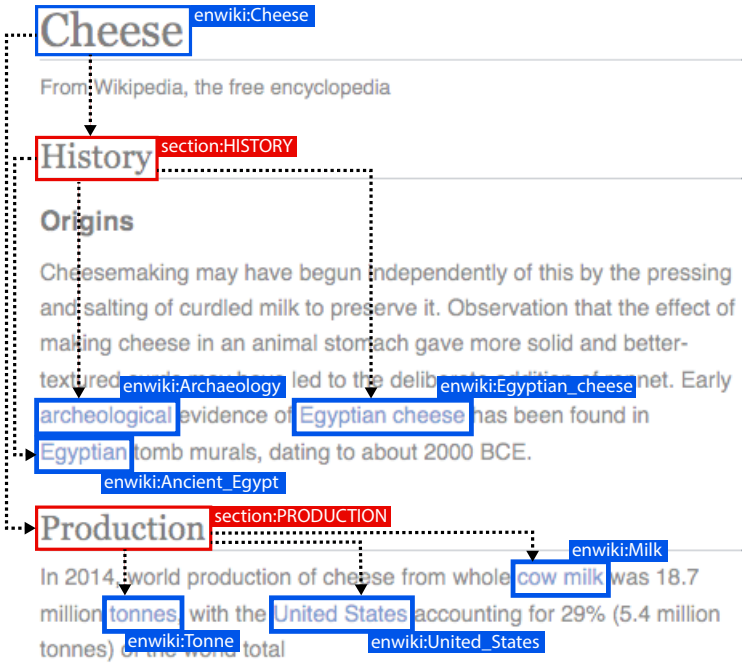

Figure 2.5 Example graph construction strategy from a Wikipedia article excerpt. Blue boxes are nodes representing the corresponding entity, and red boxes are edge labels. Adapted from `https://en.wikipedia.org/wiki/Cheese`.

in individual term being close to many topically-similar terms [92]. Within the context of CAR, we are only concerned with topics that are similar given a particular context (i.e., the question facet). Furthermore, we know that the most salient contextual information comes from entity mentions. Since article titles also correspond to entities, and headings can be considered the context, we use the following approach employing knowledge graph embeddings.

Before we can train these embeddings, we must have a knowledge graph. Since the evaluation topics we use come from Wikipedia, we cannot use a freely-available knowledge graph based on all of Wikipedia data (e.g., Freebase or DBPedia)—this could

artificially improve results by including information from the relevant paragraphs themselves.[16] Instead, we construct our own knowledge graph from the available CAR training data. Our approach is based on the observation made by WikiLinks [188]: a knowledge graph can be constructed using links in Wikipedia articles between the article's entity and entities that the page links to. Unlike WikiLinks, we enrich our knowledge graph by labeling the edges using heading information. This contextualizes the relations between entities with respect to CAR, a quality that existing resources do not provide.

More formally, let knowledge graph $G = (E, R)$, where E is the set of entities, and R is the set of relations. Let E be of the union of the set of all article topic entity and the set of all entities found in links. The set of directed relations R is defined as any pair of entities for (t, m) where t is an article topic, and m is an entity mention in a paragraph relevant to the query. The edges are labeled using the highest-level (non-title) heading of the paragraph. An illustration of this process is shown in Figure 2.5. To address the data sparsity problem caused by the large number of low-frequency headings, we only use the lemmatized syntactic head of the target heading. Furthermore, we combine any edge label that doesn't appear in the e_{max} most frequent headings into a single edge label. By using this approach, we are able to encode entity relations in a way that maintains relations between high-frequency headings—precisely the headings in most need of entity contextual hints.

We use the knowledge graph G to construct HolE embeddings for the entities and relations. HolE embeddings have been shown to be effective at entity link prediction by capturing rich interactions [133]. HolE embeddings produce a collection of entity embeddings and relation embeddings that provide similarity via circular correlation.

[16] We also cannot remove the evaluation topics when training the graph because that defeats the purpose; without target entities encoded in the embeddings, there is no way to find similar entities when ranking.

First, we collect all entity mentions in the paragraph being ranked. We us entities from links that appear in Wikipedia paragraphs that target other articles. However, because Wikipedia guidelines suggest only linking an entity on its first occurrence in an article[17], we also explore using an entity extractor to find entity mentions (DBPedia Spotlight [39]). For each entity mention, we calculate the entity similarity score, given the current query topic entity and the current heading label. We include the top $n_{entscores}$ from the paragraph during the combination phase of the model (see Figure 2.3d). This is similar to how some models (e.g., PACRR) perform k-max pooling of query term results. This approach differs from contextual vectors because these signals refer to the entire query-document combination, and not specific query terms.

Implementation Details

We apply our methods for altering the generalized interaction-focused neural ranking model to the PACRR [75] and K-NRM [214] models. Both of these models are described in Section 2.1, and have been demonstrated to be a strong approaches for ad-hoc retrieval.

For PACRR, we include contextual vectors after the query term pooling. The vectors are simply concatenated alongside the query term matching scores. For heading independence, we split out the work up through the query term pooling. That is, we perform the CNN and pooling independently by heading position. We add an additional dense layer here to further consolidate heading position information before the final combination phase. We include knowledge graph embedding scores in the combination phase, alongside the matching scores and contextual vectors.

[17] `https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style/Linking`

Table 2.6 Dataset characteristics from the CAR v1.5 data release.

			Relevance	
Dataset	Articles	Queries	Auto.	Man.
`train.fold1-2` (train)	114,387	873,746	$2.2M$	-
`test200` (validation)	198	1,860	$4.7k$	-
`benchmarkY1test` (test)	133	2,125	$5.8k$	$30k$

Includes counts of automatic (Auto.) and Manual (Man.) relevance judgments.

For K-NRM, we include contextual vectors alongside the kernel scores before combination. We split out kernel matching by heading position to achieve heading independence. Knowledge graph embedding scores are concatenated to the matching scores prior to combination.

2.3.3 Experiment

In this section, we describe our primary experiment using the approaches described in Section 2.3.2 and present our results using the CAR dataset.

Experimental Setup

Dataset We use the official TREC CAR dataset (version 1.5) for both training and evaluating our approach [45, 47]. This dataset was constructed from Wikipedia articles that represent topics (that is, it does not include meta or talk pages). The main datasource for retrieval consists of all the paragraphs, disassociated with their articles and surrounding content (30M paragraphs, `paragraphcorpus`). The dataset also includes queries, which were automatically generated from the article structure.

Table 2.7 Manual relevance judgment counts and occurrence frequencies for the CAR test dataset, benchmarkY1test.

Relevance Label	Count	%	Value
Must be mentioned	2,461	8.3%	3
Should be mentioned	1,970	6.7%	2
Can be mentioned	3,094	10.5%	1
Roughly on topic	9,219	31.2%	0
Non-relevant	12,785	43.2%	-1
Trash	42	0.1%	-2

For each query, the dataset also provides automatic relevance judgments based on the assumption that paragraphs under a particular heading are all valid answers to the corresponding query.

The dataset is split into subsets suitable for training and testing of systems (summary in Table 2.6). For a randomly-selected half of Wikipedia, all data are provided for training (split into 5 folds, `train.fold0-4`). We use folds 1 and 2 for training our models. A subset of approximately 200 articles from `train.fold0` is an evaluation dataset provided by [129] (`test200`). Finally, 133 articles from the second half of Wikipedia (the half that was not designated for training) serve as evaluation articles (`benchmarkY1test`). The TREC CAR 2017 task released manual relevance judgments in addition to the automatic judgments available for the other datasets [47]. The manual relevance judgments are graded on a scale from *Must be mentioned* (3) to *Trash* (-2). We provide a summary of the frequency of these labels in Table 2.7. While the manual relevance judgments are considered gold standard and are capable of matching relevant paragraphs from other articles, they only cover a subset of queries (702 of the 2,125 queries).

Knowledge graph embeddings We first generate a knowledge graph using the method described in Section 2.3.2. We use the entire `train.fold0-4` dataset to crease as extensive of a graph as possible. We generate two versions of the graph: one using hyperlinks as entity mentions, and one using entity mentions extracted using the automatic entity extractor DBPedia Spotlight [39] with a confidence setting of 0.5. Although this tool is less accurate than manually-created links, it captures entities that are not linked (e.g., the Wikipedia guidelines suggest only linking the first mention of an entity in an article, leaving out subsequent mentions from the graph). Edge labels are limited to only the $e_{max} = 1000$ most frequently-used labels. We test both versions of the graph when evaluating the performance of using knowledge graph embedding scores. We set the embedding length to 100, and train on the link graph using the pairwise stochastic trainer with random sampling for 5,000 training iterations (we found this to be enough iterations for the training to converge). When picking which entity scores to include in the model, we use top $n_{sent\_scores} = 2$ similarities (this matches the document term pooling parameter k in PACRR).

Training and evaluation We train and evaluate several variations of the PACRR and K-NRM models to explore the effectiveness of each approach. For both models, we use the model configurations proposed in [75] and [214], with some modifications to better suit the task. We increase the maximum query length to 18 to accommodate the longer queries often found in the CAR dataset, while shortening the maximum document length to 150 to reduce processing time (the CAR paragraphs are usually much shorter than the documents found in ad-hoc retrieval). For the PACRR model, we use an extended set of kernel sizes: up to 5×5. We made no changes to the Gaussian kernels for K-NRM proposed in [214]. We train the PACRR model for 80

iterations of 2048 samples on `train.fold1-2`, and K-NRM for 200 iterations. We found that this was long enough for each of the models to converge.

Automatic relevance judgments serve as a source of relevant documents, and we use the top non-relevant BM25 documents as negative training examples. Negative samples are used for the pairwise loss function used to train PACRR, and BM25 results offer higher-quality negative samples than random paragraph would (e.g., these examples have matching terms, whereas random paragraphs likely would not).[18] For each positive sample, we include 6 negative samples. To a point, including more negative samples has been shown to improve the performance of PACRR at the expense of training time; we found 6 negative samples to be an effective balance between the two considerations. The training iteration that yields the highest R-Precision value on the validation dataset (`test200`) is selected for evaluation on the test dataset. We then rerank the top 100 BM25 results for each query in `benchmarkY1test`, and test using the manual and automatic relevance judgments. For each configuration, we report the 4 official TREC CAR metrics: Mean Average Precision (MAP), R-Precision (R-Prec), Mean Reciprocal Rank (MRR), and normalized Discounted Cumulative Gain (nDCG). We compare the results to an unmodified version of PACRR trained using the same approach, the initial BM25 ranking, and the other top approaches submitted to TREC.

[18]We acknowledge that some paragraphs included as negative training samples, if inspected manually, would be found relevant due to the limitations of the automatic relevance judgments. We deem this as okay, considering the high occurrence of non-relevant documents in the manual relevance judgments, and the comparatively poor performance of BM25 at CAR.

Table 2.8 CAR performance results under various models.

Approach	MAP	R-Prec	MRR	nDCG
Manual (including unjudged)				
PACRR (no modification)	0.208	0.219	0.445	0.403
+ CV*	▲ **0.211**	**0.221**	**0.453**	▲ **0.408**
+ HI	0.205	0.213	0.442	0.403
+ HI + CV	0.204	0.214	0.440	0.401
+ HI + CV + KG (links)	▼ 0.198	▼ 0.206	0.429	0.395
+ HI + CV + KG (extr.)	0.200	▼ 0.206	0.433	0.396
K-NRM (no modification)	0.161	0.170	0.346	0.350
+ CV	△ 0.171	0.176	0.354	△ 0.360
+ HI	△ 0.192	△ 0.200	△ 0.414	△ 0.386
+ HI + CV	△ 0.194	△ 0.203	△ 0.411	△ 0.388
+ HI + CV + KG (links)	△ 0.193	△ 0.201	△ 0.411	△ 0.387
+ HI + CV + KG (extr.)	△ 0.192	△ 0.203	△ 0.408	△ 0.384
Sequential dependence model* [100]	▼ △ 0.172	▼ △ 0.186	▼ △ 0.393	▼ 0.350
Siamese attention network* [122]	▼ ▽ 0.137	▼ 0.171	▼ 0.345	▼ ▽ 0.274
BM25 baseline*	▼ ▽ 0.138	▼ 0.158	▼ 0.317	▼ ▽ 0.296
Manual (excluding unjudged)				
PACRR (no modification)	0.471	0.496	0.774	0.583
+ CV*	▲ 0.480	▲ **0.509**	▲ **0.794**	▲ 0.592
+ HI	▲ 0.479	▲ 0.505	▲ 0.795	▲ 0.593
+ HI + CV	▲ 0.479	0.499	0.782	▲ 0.590
+ HI + CV + KG (links)	▲ 0.482	▲ 0.505	0.787	▲ 0.592
+ HI + CV + KG (extr.)	▲ **0.485**	▲ 0.505	▲ 0.792	▲ **0.594**
K-NRM (no modification)	0.446	0.474	0.732	0.556
+ CV	△ 0.453	0.481	0.732	0.562
+ HI	△ 0.470	△ 0.492	△ 0.771	△ 0.578
+ HI + CV	△ 0.473	△ 0.497	△ 0.774	△ 0.582
+ HI + CV + KG (links)	△ 0.467	△ 0.493	△ 0.755	△ 0.577
+ HI + CV + KG (extr.)	△ 0.469	△ 0.490	△ 0.763	△ 0.577
BM25 baseline*	▼ 0.452	▼ 0.476	▼ △ 0.747	▼ 0.566

Manual is evaluated both including and excluding documents without relevance judgments. The top value in each section is in bold. Records marked with * were included in the TREC document pool. Significant results compared to the unmodified PACRR and K-NRM models within each section are marked with ▲/▼ and △/▽, respectively (paired t-test, 95% confidence). Baselines are compared with both both models. CV: contextual vector. HI: heading independence. KG: knowledge graph. links: entities from Wikipedia links. extr.: automatic entity extraction.

Table 2.9 CAR performance results under various models with automatic relevance judgments.

Approach	MAP	R-Prec	MRR	nDCG
Automatic				
PACRR (no modification)	0.164	0.131	0.247	0.254
+ CV*	▲ 0.170	0.134	▲ 0.255	▲ 0.259
+ HI	▲ 0.171	0.139	▲ 0.256	▲ 0.260
+ HI + CV	**▲ 0.176**	**▲ 0.146**	**▲ 0.263**	**▲ 0.265**
+ HI + CV + KG (links)	▲ 0.174	▲ 0.141	▲ 0.260	▲ 0.263
+ HI + CV + KG (extr.)	▲ 0.173	▲ 0.140	▲ 0.260	▲ 0.262
K-NRM (no modification)	0.117	0.086	0.177	0.209
+ CV	△ 0.129	△ 0.095	△ 0.191	△ 0.221
+ HI	△ 0.156	△ 0.125	△ 0.234	△ 0.246
+ HI + CV	△ 0.158	△ 0.124	△ 0.235	△ 0.247
+ HI + CV + KG (links)	△ 0.156	△ 0.121	△ 0.233	△ 0.245
+ HI + CV + KG (extr.)	△ 0.154	△ 0.124	△ 0.230	△ 0.243
Sequential dependence model* [100]	▼ △ 0.130	▼ △ 0.101	▼ △ 0.196	▼ 0.207
Siamese attention network* [122]	▼ 0.105	▼ 0.083	▼ 0.161	▼ ▽ 0.152
BM25 baseline*	▼ 0.106	▼ 0.085	▼ 0.159	▼ ▽ 0.170

The top value in each section is in bold. Records marked with * were included in the TREC document pool. Significant results compared to the unmodified PACRR and K-NRM models within each section are marked with ▲/▼ and △/▽, respectively (paired t-test, 95% confidence). Baselines are compared with both both models. CV: contextual vector. HI: heading independence. KG: knowledge graph. links: entities from Wikipedia links. extr.: automatic entity extraction.

Results

We present the performance of our methods in Tables 2.8 and 2.9. Overall, the results show that our methods perform favorably compared to the unmodified PACRR and K-NRM models, the other top submissions to TREC CAR 2017 (sequential dependency model [100] and the Siamese attention network [122]), and the BM25 baseline (which our method re-ranks).

Due to the relatively low number of manual relevance judgments per query,[19] we report manual relevance judgments both including and excluding unjudged paragraphs. When unjudged paragraphs are included, they are assumed to be irrelevant.[20] When unjudged documents are excluded, we filter unjudged paragraphs from the ranked lists (i.e., we perform a condensed list evaluation). The condensed list evaluation is a better comparison for methods that were not included in the evaluation pool [173]. Indeed, PACRR with the heading position and heading frequency contextual vectors outperforms the all other approaches, and significantly outperforms the unmodified PACRR model in terms of MAP and nDCG. Interestingly, the knowledge base approaches perform significantly worse than the unmodified PACRR in terms of R-Prec when unjudged paragraphs are included. However, when unjudged paragraphs are not included, it performs significantly better in every case. In fact, the version that uses entity extraction when calculating entity scores performs best overall in terms of MAP and nDCG. This shows that these approaches are effective, and likely rank paragraphs that are relevant yet unjudged high (reducing the score when unjudged documents are included).

[19] On average, there are 42 manual relevance judgments for the 702 queries that were manually assessed.

[20] Unjudged evaluation is unavailable for the sequential dependency model and Siamese attention network.

When considering automatic relevance judgments, heading independence and the heading frequency contextual vector significantly outperforms unmodified PACRR, and performs best overall in every metric. By R-Prec, this configuration outperforms the next best approach (sequential dependence model) by 26%. However, the performance when evaluating with manual judgments does not significantly outperform unmodified PACRR. Since training is also conducted using automatic relevance judgments, this may be caused by PACRR overfitting to this sense of relevance. Interestingly, when the knowledge graphs features are added to this approach, performance drops with automatic judgments, but increases with manual judgments. This suggests that these features are useful for determining human-classified relevance.

Although the K-NRM does not outperform the PACRR model in our experiments, it is worth noting that our approach yields significant improvements for K-NRM compared to the unmodified model. Specifically, contextual vectors yield significant improvements when evaluating using both with manual and automatic relevance judgments three environments. Beyond that, using heading independence significantly improves on the contextual vector results. This indicates that our approaches are generally applicable to interaction-based neural ranking models for CAR.

2.3.4 Analysis

In this section, we investigate several questions surrounding the results to gain more insights into the behavior and functionality of the approaches to CAR detailed in Section 2.3.2.

Characteristics of Heading Positions

Our approaches assert that heading positions (i.e., title, intermediate, and target heading) act as a signal for heading utility. In Section 2.3.3 we showed that including

the heading position as either the heading position contextual vector or via and heading independence improves performance beyond the baseline neural architecture. Here, we test the hypothesis that terms found in different heading positions exhibit different behaviors in relevant documents.

We use the *term occurrence rate* to assist in this analysis. Let the term occurrence rate $occ(h)$ for a given heading h be the probability that any term in a given heading appears in relevant paragraphs. More formally:

$$occ(h) = \frac{\sum_{p \in rel(h)} I(\exists t \in p | t \in h)}{|rel(h)|}$$

where $rel(h)$ is the set of relevant paragraphs for p and I is the indicator function. Although the term occurrence rate only accounts for a single binary term match within relevant documents, this assumption is justified by the fact that headings are usually terse, and the PACRR model only considers the top 2 matches for each query term for ranking purposes (query term pooling). Here, we use the term occurrence rate as a proxy for heading utility.

In Figure 2.6, we plot a kernel density estimation of term occurrence distributions for each heading position. We use all topics from the training dataset, and calculate term occurrence rates using automatic relevance judgments. The figure shows that target headings are much more likely to appear in relevant documents than titles and intermediate headings, with a much higher density at the term occurrence rate of 1. This matches our prediction that target headings are more likely to be topical, and therefore appear in relevant documents. Furthermore, the distributions of intermediate and title headings are roughly opposite each other, with titles more likely to occur than intermediate headings. Note that, due to the hierarchical nature of CAR queries, intermediate headings also appear as target headings in other queries for the same topic. This contributes to the high frequency of target headings with a term

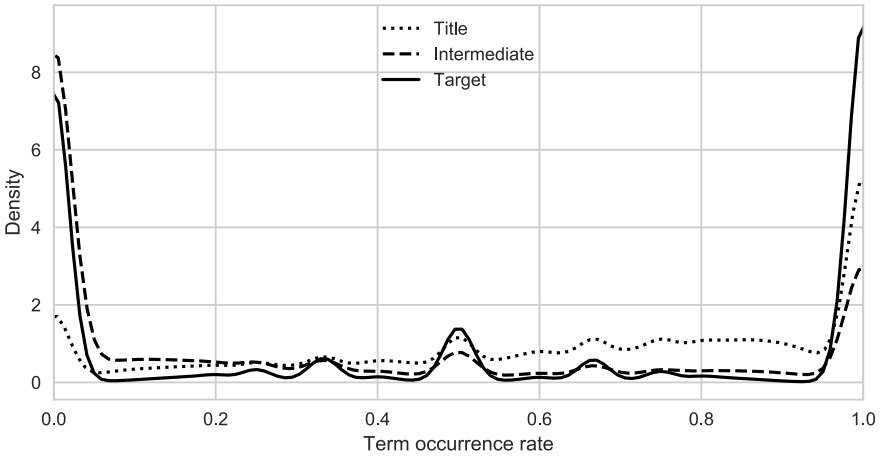

Figure 2.6 Kernel density estimation for target (solid), intermediate (dashed), and title (dotted) heading term occurrence rates Results are based on automatic judgments in train.fold0.

occurrence rate of 0. Furthermore, many target headings are only used in a single article (i.e., only appear once in the Wikipedia collection), with only a handful of paragraphs associated with them. This explains the multi-modal distribution seen for target headings; the small denominators result in local maxima near $\frac{1}{3}$, $\frac{1}{2}$, and $\frac{2}{3}$.

Characteristics of High-frequency Headings

In Section 2.3.3, we showed that including the heading frequency contextual vector improved retrieval performance. We predicted that the model is able to use this information to predict which query terms are likely to appear in relevant documents. To investigate this behavior, we look at whether heading frequency is correlated with term occurrence rates, and whether there is a performance gap between low- and high-frequency headings.

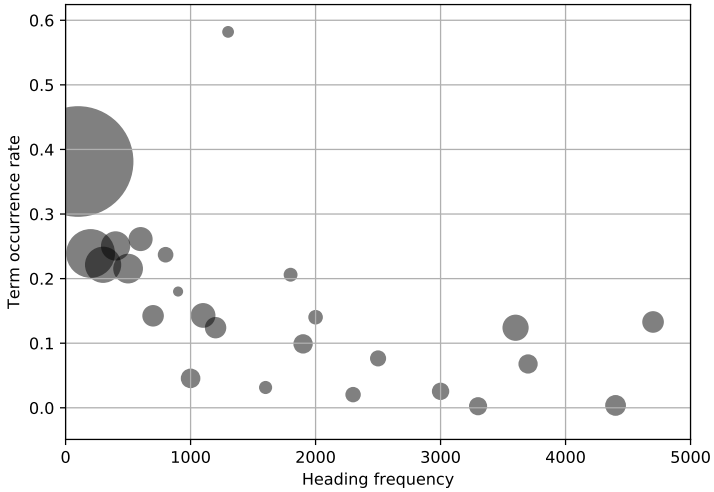

Figure 2.7 Term occurrence rate plotted by heading frequency. Heading frequency is grouped and averaged by 100 for clarity. The area of each point is proportional to the number of heading instances used to calculate the term occurrence rate. One very high frequency heading (History) was omitted for readability (heading frequency: 15,220, term occurrence rate: 0.035).

In Figure 2.7, we plot term occurrence rates by heading frequency. The figure only includes headings that occur at least twice in `train.fold0`. We remove the extremely-frequent heading 'History' and bin frequencies by 100 to improve readability. The area of each point is proportional to the number of heading instances in the bin. It is clear from the figure that the less frequent a heading is, the higher the term occurrence rate. In general, headings with a frequency less than 1,000 have an average heading frequency 10-20 points higher than higher-frequency headings. This indicates that heading frequency is a valuable signal for estimating the utility of a heading during ranking.

Table 2.10 MAP scores stratified by heading frequency of target heading for each query

Model	Infrq.	0-20%	20-40%	40-60%	60-80%	80-100%
PACRR (unmod.)	0.503	0.521	0.453	0.388	0.412	0.366
+ HP	0.509	0.520	0.464	0.400	**0.430**	0.377
+ HP + HF	0.511	0.519	0.470	0.406	0.417	0.383
+ HI	0.509	0.525	**0.479**	0.382	0.417	**0.391**
+ HI + HF	0.509	0.519	0.474	0.403	0.409	0.379
+ HI + HF + KG (links)	0.511	0.529	0.475	0.412	0.408	0.376
+ HI + HF + KG (extr.)	**0.513**	**0.535**	0.477	**0.416**	0.407	0.380
BM25	0.482	0.480	0.429	0.378	0.420	0.373
Query count	349	89	96	81	43	44
Max. % increase (to unmod.)	+2.0%	+2.7%	+5.7%	+7.2%	+4.3%	+6.8%

Uses manual relevance judgments, excluding unjudged. Infrequent headings that do not appear in the training dataset are included in the 'Infrq.' column. The approaches yield the highest performance improvement on the high-frequency (low-utility) queries.

Difficultly Matching High-frequency Headings

Given the knowledge that high-frequency headings generally exhibit a low term occurrence rate, we are interested in measuring whether there exists a discrepancy between the performance at different heading frequencies. Table 2.10 shows a MAP[21] performance breakdown on the test dataset (manual relevance judgments, unjudged excluded) stratified by the heading frequency of the target heading found in the training dataset. The number of queries in each stratum varies because the strata were selected from equally-spaced breakpoints in the training data, and the test set does not represent a uniform selection of Wikipedia articles. The most frequent head-

[21]We observed similar behavior for MAP, R-Prec, MRR, and nDCG, so we only report MAP here.

ings (80-100%, e.g., 'History' and 'Early Life') exhibit the worst performance among all strata. This matches our intuition that these low-utility headings are difficult to match. The low frequency headings exhibit the highest performance by all models. Notice that the gains compared to the unmodified PACRR model are higher for high-frequency headings than the low-frequency headings, despite having a worse performance overall. This validates our claims that low-utility headings are harder to match than high-utility headings, and that our approaches are able to improve performance for these queries.

Qualitative Query Analysis

To gain a more thorough understanding of how our methods enhance ranking techniques for CAR, we perform qualitative analysis. Table 2.11 gives example answers and rankings by various configurations. Query 1 in the table (Instant Coffee » History) saw an improvement from a MAP of 0.4468 (PACRR no modification) to 0.6987 (PACRR + HI + CV + KG (extr.)) when considering manually-judged answers. Both the unmodified model and the model with heading independence ranked the highly-relevant paragraph low (rank 10 and 9), and a non-relevant answer high (rank 1), while the knowledge graph approach did the reverse (rank 1 and 18, respectively). This is likely due to the difference in entities encountered in the answer. The relevant answer has historic and political entities such as *World War II* and the *United Kingdom*, while the non-relevant answer includes artists such as *Jenifer Papararo*.

We notice that in other situations, however, the knowledge base approach fails. In Query 2 of Table 2.11 (Taste » Basic tastes » Sweetness), the unmodified PACRR model performs better than all variations we explore, with a MAP of 0.7444, compared to 0.4221 for the knowledge graph approach. In this case, the top-level heading is not common enough to be included in the knowledge graph embeddings, so it

Table 2.11 Concrete ranking examples in case where knowledge graph method work well (Query 1), and case where knowledge graph method does not work well (Query 2).

Relevance	Answer	Rank (no mod.)	Rank (HI)	Rank (extr.)
	Query 1: Instant Coffee » History			
3 (must)	In some countries, such as Portugal, Spain, and India, instant coffee is commonly mixed with hot milk instead of boiling water. In other countries, such as South Korea, instant coffee commonly comes pre-mixed with non-dairy creamer and sugar and is called "coffee mix". Said to have been popularised in the UK by GIs during World War II, instant coffee still accounts for over 75 percent of coffee bought to drink in British homes, as opposed to well under 10 percent in the U.S. and France and one percent in Italy.	10	9	1
-1 (non-rel)	"Instant Coffee" is a Canadian artist collective based in Vancouver and Toronto. Formed in 2000, the collective's membership has undergone a number of changes. Instant Coffee's most active members have included Cecilia Berkovic, Jinhan Ko, Kelly Lycan, Jenifer Papararo, and Khan Lee.	1	1	18
	Query 2: Taste » Basic tastes » Sweetness			
3 (must)	Sweetness, usually regarded as a pleasurable sensation, is produced by the presence of sugars and a few other substances. Sweetness is often connected to aldehydes and ketones, which contain a carbonyl group. Sweetness is detected by a variety of G protein coupled receptors coupled to the G protein gustducin found on the taste buds. At least two different variants of the "sweetness receptors" must be activated for the brain to register sweetness. Compounds the brain senses as sweet are thus compounds...	5	4	11
0 (topic)	"Umami", or "savory" taste, is one of the five basic tastes (together with sweetness, sourness, bitterness, and saltiness). It has been described as brothy or meaty.	9	13	6

The ranks are provided by the PACRR model with: no modification (no mod.); heading independence (HI); and heading independence, heading frequency, and knowledge graph using automatic entity extraction (extr.).

was collapsed into the remainder class. This explains why it ranked the non-relevant answer listing other tastes high, while pushing down the relevant answer that includes more specific language pertaining to the reception of sweetness. This demonstrates the need in future work to better address low-frequency headings.

2.3.5 Summary

In this section, we proposed an approach for utilizing the characteristics of a dataset (namely, the CAR dataset) to improve neural semantic ranking performance. Specifically, we incorporated information about the structured query, contextual information about the frequency of query components, and entity mentions within the query and text.

2.4 Discussion and Conclusions

In this chapter, I first demonstrated that neural rankers can be effectively trained using large sources of naturally-occurring text pairs, such as headline-article pairs from news articles. This approach can yield ranking models that significantly outperform a tuned baseline model, models trained using the prior best weak supervision technique, and even models trained using in-domain data. This validates Hypothesis 1.1. I then demonstrated that neural ranking architectures can easily incorporate task-specific dataset characteristics. I showed this using the TREC Complex Answer Retrieval dataset. Neural rankers for this task benefit from characteristics about the structured query and entity mentions in two leading neural ranking architectures. This validates Hypothesis 1.2.

These works were done prior to the widespread availability and adoption of large-scale contextualized language models, such as BERT [44]. In the following chapter, I

demonstrate that these models are effective for ranking. As such, I re-visit the topic of transferring relevance signals across datasets in the context of contextualized language models, further validating Hypothesis 1.1. I also explore how dataset statistics can be used to improve the training process with contextualized language models, further validating Hypothesis 1.2.

Chapter 3

Ranking Effectiveness of Neural Models with Contextualization

It is no exaggeration to suggest that large-scale pre-trained contextualized language models such as BERT [44] have been transformative in the field of NLP. These models take advantage of massive amounts of text to build contextualized word representations, i.e., representations that consider the surrounding words. These models can be effectively fine-tuned to a variety of tasks. Many soon observed that these models are beneficial for the task of ad-hoc ranking [135, 156, 220]. In this section, I present my contributions in this area, which include (1) showing that these models can be incorporated into prior neural ranking architectures, (2) that their benefits for ranking extend beyond ad-hoc search, (3) that these models can allow for effective transfer of relevance signals across natural languages and domains, and (4) that incorporating dataset statistics in the training process can improve effectiveness.

The remainder of this chapter is organized as follows. First, I present background information in Section 3.1. I then show how contextualized language models can be effectively incorporated into existing neural ranking models in Section 3.2, validating Hypothesis 1.3. I then move beyond ad-hoc ranking in Section 3.3 and show that these models can also be effective in another ranking task: ranking of clinical report versions by the degree of discrepancy. This further validates Hypothesis 1.3. In Section 3.4, I show that contextualized language models allow for the effective transfer of relevance signals to languages other than English with limited training data, validating Hypothesis 1.4. I then re-visit Hypotheses 1.1 and 1.2 by showing that dataset

statistics can be incorporated into the training process to improve effectiveness (Section 3.5) and that relevance signals can be transferred across tasks (Section 3.6). Finally, in Section 3.7, I recap the conclusions of this chapter.

3.1 Background and Preliminaries

Static word embeddings (i.e., word representations) are inherently limited by the fact that they only represent a single word. While in some cases this is appropriate, in many others this can yield unsatisfactory results. For instance, the term *bats* has many senses with considerably different meanings: nocturnal mammals, long wooden implements, and a fluttering motion, to name a few. In many cases, the sense can be reasonably disambiguated by the surrounding context, e.g., "the *bats* flew right past me", "the baseball *bats* are there in the bin", and "she *bats* her eyes". This intuition lead many to pursue building word representations that incorporate the surrounding context, which has generally been successful in improving downstream NLP tasks.

Early work in this area, such as ECO embeddings [48], focused on building contextualized representations of n-grams by better modeling composition. For the purposes of ranking, this is conceptually similar to ConvKNRM, which builds models of n-grams (albeit via a convolutional neural network). Peters et al. presented a contextualized language model built from a multi-layer bi-directional recurrent neural network (ELMo), and demonstrated that it can simply replace the embedding layer of other models [153]. This model is trained to predict missing terms in sentences as a pre-training task. This yields embeddings that consistently improve the effectiveness of various natural language processing tasks. However, it is limited by the underlying bi-directional recurrent structure both in time (expensive to run for long sequences), and in context (each direction can only provide context from that side).

Devlin et al. address some of these issues in BERT, which applies a similar idea but uses a self-attentive [198] transformer network instead of a recurrent network [44]. This work also introduced a new pre-training objective in addition to masked language modeling: next sentence prediction. This objective allows the pre-trained model to be easily tuned for tasks the operate on text pairs. For information retrieval, this has been very successful when applied to query and document pairs [135, 156, 220]. Due to the wild success of BERT on various NLP tasks, others have investigated alternative pre-training settings and architectures, notably Transformer-XL [38], XLNet [222], XLM [89], RoBERTa [105], and ALBERT [90]. Other models focus on textual generation, such as GPT-2 [158], and T5 [159].

3.2 Effective Ranking using Contextualized Language Models

With the availability of pretrained contextualized language models, we wondered whether these resources would be more effective than the static word embedding representations used in prior work for ranking. In this section, we examine the utilization of pretrained contextualized term representations on ad-hoc document ranking.

3.2.1 Methodology

Contextualized Similarity Tensors

Pretrained contextual language representations (such as those from ELMo [153] and BERT [44]) are context sensitive; in contrast to more conventional pretrained word vectors (e.g., GloVe [151]) that generate a single word representation for each word in the vocabulary, these models generate a representation of each word based on its context in the sentence. For example, the contextualized representation of word *bank* would be different in *bank deposit* and *river bank*, while a pretrained word embedding

model would always result in the same representation for this term. Given that these representations capture contextual information in the language, we investigate how these models can also benefit general neural ranking models.

Although contextualized language models vary in particular architectures, they typically consist of multiple stacked layers of representations (e.g., recurrent or transformer outputs). The intuition is that the deeper the layer, the more context is incorporated. To allow neural ranking models to learn which levels are most important, we choose to incorporate the output of all layers into the model, resulting in a three-dimensional similarity representation. Thus, we expand the similarity representation (conditioned on the query and document context) to $\mathbf{S_{Q,D}} \in \mathbb{R}^{L \times |Q| \times |D|}$ where L is the number of layers in the model, akin to the channel in image processing. Let $context_{\mathbf{Q},\mathbf{D}}(t, l) \in \mathbb{R}^D$ be the contextualized representation for token t in layer l, given the context of Q and D. Given these definitions, let the contextualized representation be:

$$\mathbf{S_{Q,D}}[l, q, d] = cos(context_{\mathbf{Q},\mathbf{D}}(q, l), context_{\mathbf{Q},\mathbf{D}}(d, l)) \tag{3.1}$$

for each query term $q \in Q$, document term $d \in D$, and layer $l \in [1..L]$. Note that when q and d are identical, they will likely not receive a similarity score of 1, as their representation depends on the surrounding context of the query and document. The layer dimension can be easily incorporated into existing neural models. For instance, soft n-gram based models, like PACRR, can perform convolutions with multiple input channels, and counting-based methods (like KNRM and DRMM) can count each channel individually.

Joint BERT Approach

Unlike ELMo, the BERT model encodes multiple text segments simultaneously, allowing it to make judgments about text pairs. It accomplishes this by encoding two

meta-tokens (`[SEP]` and `[CLS]`) and using text segment embeddings (*Segment A* and *Segment B*). The `[SEP]` token separates the tokens of each segment, and the `[CLS]` token is used for making judgments about the text pairs. During training, `[CLS]` is used for predicting whether two sentences are sequential – that is, whether *Segment A* immediately precedes *Segment B* in the original text. The representation of this token can be fine-tuned for other tasks involving multiple text segments, including natural language entailment and question answering [219].

We explore incorporating the `[CLS]` token's representation into existing neural ranking models as a joint approach. This allows neural rankers to benefit from deep semantic information from BERT in addition to individual contextualized token matches.

Incorporating the `[CLS]` token into existing ranking models is straightforward. First, the given ranking model produces relevance scores (e.g., n-gram or kernel scores) for each query term based on the similarity matrices. Then, for models using dense combination (e.g., PACRR, KNRM), we propose concatenating the `[CLS]` vector to the model's signals. For models that sum query term scores (e.g., DRMM), we include the `[CLS]` vector in the dense calculation of each term score (i.e., during combination of bin scores).

We hypothesize that this approach will work because the BERT classification mechanism and existing rankers have different strengths. The BERT classification benefits from deep semantic understanding based on next-sentence prediction, whereas ranking architectures traditionally assume query term repetition indicates higher relevance. In reality, both are likely important for relevance ranking.

3.2.2 Experiment

Experimental Setup

Datasets. We evaluate our approaches using two datasets: Trec Robust 2004 and WebTrack 2012–14. For Robust, we use the five folds from [77] with three folds used for training, one fold for testing, and the previous fold for validation. For WebTrack, we test on 2012–14, training each year individually on all remaining years (including 2009–10), and validating on 2011. (For instance, when testing on WebTrack 2014, we train on 2009–10 and 2012–13, and validate on 2011.) Robust uses Trec discs 4 and 5[1], WebTrack 2009–12 use ClueWeb09b[2], and WebTrack 2013–14 uses ClueWeb12[3] as document collections. We evaluate the results using the nDCG@20 / P@20 metrics for Robust04 and nDCG@20 / ERR@20 for WebTrack.

Models. Rather than building new models, in this work we use existing model architectures to test the effectiveness of various input representations. We evaluate our methods on three neural relevance matching methods: PACRR [76], KNRM [214], and DRMM [56]. Relevance matching models have generally shown to be more effective than semantic matching models, while not requiring massive amounts of behavioral data (e.g., query logs). For PACRR, we increase $k_{max} = 30$ to allow for more term matches and better back-propagation to the language model.

Contextualized language models. We use the pretrained ELMo (Original, 5.5B) and BERT (BERT-Base, Uncased) language models in our experiments. For ELMo, the query and document are encoded separately. Since BERT enables encoding multiple texts at the same time using *Segment A* and *Segment B* embeddings, we encode the query (*Segment A*) and document (*Segment B*) simultaneously. Because

[1] 520k documents; `https://trec.nist.gov/data_disks.html`
[2] 50M web pages, `https://lemurproject.org/clueweb09/`
[3] 733M web pages, `https://lemurproject.org/clueweb12/`

the pretrained BERT model is limited to 512 tokens, longer documents are split such that document segments are split as evenly as possible, while not exceeding the limit when combined with the query and control tokens. (Note that the query is always included in full.) BERT allows for simple classification fine-tuning, so we also experiment with a variant that is first fine-tuned on the same data using the Vanilla BERT classifier (see baseline below), and further fine-tuned when training the ranker itself.

Training and optimization. We train all models using pairwise hinge loss [43]. Positive and negative training documents are selected from the query relevance judgments (positive documents limited to only those meeting the re-ranking cutoff threshold k using BM25, others considered negative). We train each model for 100 epochs, each with 32 batches of 16 training pairs. Gradient accumulation is employed when the batch size of 16 is too large to fit on a single GPU. We re-rank to top k BM25 results for validation, and use P@20 on Robust and nDCG@20 on WebTrack to select the best-performing model. We different re-ranking functions and thresholds at test time for each dataset: BM25 with $k = 150$ for Robust04, and QL with $k = 100$ for WebTrack. The re-ranking setting is a better evaluation setting than ranking all qrels, as demonstrated by major search engines using a pipeline approach [170]. All models are trained using Adam [86] with a learning rate of 0.001 while BERT layers are trained at a rate of 2e-5.[4] Following prior work [76], documents are truncated to 800 tokens.

Baselines. We compare contextualized language model performance to the following strong baselines:

[4]Pilot experiments showed that a learning rate of 2e-5 was more effective on this task than the other recommended values of 5e-5 and 3e-5 by [44].

Table 3.1 Ranking performance of contextualized models on Robust04.

		Robust04	
Ranker	Input Representation	P@20	nDCG@20
BM25	n/a	0.3123	0.4140
SDM [125]	n/a	0.3749	0.4353
`TREC-Best`	n/a	**0.4386**	**0.5030**
ConvKNRM	GloVe	0.3349	0.3806
Vanilla BERT	BERT (fine-tuned)	[BC] 0.4042	[BC] 0.4541
PACRR	GloVe	0.3535	[C] 0.4043
PACRR	ELMo	[C] 0.3554	[C] 0.4101
PACRR	BERT	[C] 0.3650	[C] 0.4200
PACRR	BERT (fine-tuned)	[BCVG] 0.4492	[BCVG] 0.5135
CEDR-PACRR	BERT (fine-tuned)	**[BCVG] 0.4559**	**[BCVG] 0.5150**
KNRM	GloVe	0.3408	0.3871
KNRM	ELMo	[C] 0.3517	[CG] 0.4089
KNRM	BERT	[BCG] 0.3817	[CG] 0.4318
KNRM	BERT (fine-tuned)	[BCG] 0.4221	[BCVG] 0.4858
CEDR-KNRM	BERT (fine-tuned)	**[BCVGN] 0.4667**	**[BCVGN] 0.5381**
DRMM	GloVe	0.2892	0.3040
DRMM	ELMo	0.2867	0.3137
DRMM	BERT	0.2878	0.3194
DRMM	BERT (fine-tuned)	[CG] 0.3641	[CG] 0.4135
CEDR-DRMM	BERT (fine-tuned)	**[BCVGN] 0.4587**	**[BCVGN] 0.5259**

Significant improvements to [B]M25, [C]onvKNRM, [V]anilla BERT, the model trained with [G]lOve embeddings, and the corresponding [N]on-CEDR system are indicated in brackets (paired t-test, $p < 0.05$).

Table 3.2 Ranking performance on WebTrack 2012–14.

		WebTrack 2012–14	
Ranker	Input Representation	nDCG@20	ERR@20
BM25	n/a	0.1970	0.1472
SDM [125]	n/a	-	-
`TREC-Best`	n/a	0.2855	**0.2530**
ConvKNRM	GloVe	[B] 0.2547	[B] 0.1833
Vanilla BERT	BERT (fine-tuned)	**[BC] 0.2895**	[BC] 0.2218
PACRR	GloVe	0.2101	0.1608
PACRR	ELMo	[BG] 0.2324	[BG] 0.1885
PACRR	BERT	0.2225	0.1817
PACRR	BERT (fine-tuned)	[BCG] 0.3080	[BCG] 0.2334
CEDR-PACRR	BERT (fine-tuned)	**[BCVGN] 0.3373**	**[BCVGN] 0.2656**
KNRM	GloVe	[B] 0.2448	0.1755
KNRM	ELMo	0.2227	0.1689
KNRM	BERT	[B] 0.2525	[B] 0.1944
KNRM	BERT (fine-tuned)	[BCVG] 0.3287	[BCVG] 0.2557
CEDR-KNRM	BERT (fine-tuned)	**[BCVG] 0.3469**	**[BCVG] 0.2772**
DRMM	GloVe	0.2215	0.1603
DRMM	ELMo	[B] 0.2271	0.1762
DRMM	BERT	[BG] 0.2459	[BG] 0.1977
DRMM	BERT (fine-tuned)	[BG] 0.2598	[B] 0.1856
CEDR-DRMM	BERT (fine-tuned)	**[BCVGN] 0.3497**	**[BCVGN] 0.2621**

Significant improvements to [B]M25, [C]onvKNRM, [V]anilla BERT, the model trained with [G]lOve embeddings, and the corresponding [N]on-CEDR system are indicated in brackets (paired t-test, $p < 0.05$).

- BM25 and SDM [125], as implemented by Anserini [217]. Fine-tuning is conducted on the test set, representing the maximum performance of the model when using static parameters over each dataset.[5] We do not report SDM performance on WebTrack due to its high cost of retrieval on the large ClueWeb collections.
- Vanilla BERT ranker. We fine-tune a pretrained BERT model (`BERT-Base, Uncased`) with a linear combination layer stacked atop the classifier `[CLS]` token. This network is optimized the same way our models are, using pairwise cross-entropy loss and the Adam optimizer. We use the approach described above to handle documents longer than the capacity of the network, and average the `[CLS]` vectors from each split.
- `TREC-best`: We also compare to the top-performing topic TREC run for each track in terms of nDCG@20. We use `uogTrA44xu` for WT12 ([96], a learning-to-rank based run), `clustmrfaf` for WT13 ([160], clustering-based), `UDInfoWebAX` for WT14 ([104], entity expansion), and `pircRB04t3` for Robust04 ([88], query expansion using Google search results).[6]
- ConvKNRM [37], our implementation with the same training pipeline as the evaluation models.
- Each evaluation model when using GloVe [151] vectors.[7]

Results and Analysis

Tables 3.1 and 3.2 show the ranking performance using our approach. We first note that the Vanilla BERT method significantly outperforms the tuned BM25 [B] and ConvKNRM [C] baselines on its own. This is encouraging, and shows the ranking

[5] k_1 in 0.1–4 (by 0.1), b in 0.1–1 (by 0.1), SDM weights in 0–1 (by 0.05).
[6] We acknowledge limitations of the TREC experimentation environment.
[7] `glove.42B.300d`, `https://nlp.stanford.edu/projects/glove/`

power of the Vanilla BERT model. When using contextualized language term representations without tuning, PACRR and DRMM performance is comparable to that of GloVe [G], while KNRM sees a modest boost. This might be due to KNRM's ability to train its matching kernels, tuning to specific similarity ranges produced by the models. (In contrast, DRMM uses fixed buckets, and PACRR uses maximum convolutional filter strength, both of which are less adaptable to new similarity score ranges.) When fine-tuning BERT, all three models see a significant boost in performance compared to the GloVe-trained version. PACRR and KNRM see comparable or higher performance than the Vanilla BERT model. This indicates that fine-tuning contextualized language models for ranking is important. This boost is further enhanced when using the CEDR (joint) approach, with the CEDR models always outperforming Vanilla BERT [V], and nearly always significantly outperforming the non-CEDR versions [N]. This suggests that term counting methods (such as KNRM and DRMM) are complementary to BERT's classification mechanism. Similar trends for both Robust04 and WebTrack 2012–14 indicate that our approach is generally applicable to ad-hoc document retrieval tasks.

To gain a better understanding of how the contextual language model helps enhance the input representation, we plot example similarity matrices based on GloVe word embeddings, ELMo representations (layer 2), and fine-tuned BERT representations (layer 5) in Figure 3.1. In these examples, two senses of the word *curb* (restrain, and edge of street) are encountered. The first is relevant to the query (it's discussing attempts to restrain population growth). The second is not (it discusses street construction). Both the ELMo and BERT models give a higher similarity score to the correct sense of the term for the query. This ability to distinguish different senses of terms is a strength of contextualized language models, and likely can explain some of the performance gains of the non-joint models.

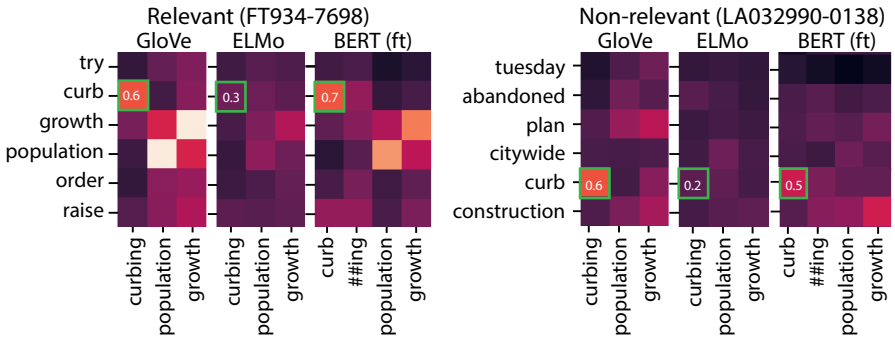

Figure 3.1 Example similarity matrix excerpts. Embedding sources include GloVe, ELMo, and BERT. Both a relevant and non-relevant document are shown. The example is for Robust query 435. Lighter values have higher similarity.

3.2.3 Summary

We demonstrated that contextualized word embeddings can be effectively incorporated into existing neural ranking architectures. This supports the hypothesis that contextualized representations are beneficial for ad-hoc ranking. We see that this approach outperforms prior ranking models. Finally, we see that even the Vanilla BERT strategy, i.e., using the classification mechanism of the contextualized language model itself, is an effective approach.

3.3 Ranking Significant Discrepancies in Clinical Reports

With the effectiveness of contextualized language models demonstrated in Section 3.2, we wondered whether these resources could benefit other IR tasks. In this section, we explored one such task: ranking the level of discrepancy of clinical reports. This presented new challenges, and led us to propose a new model for this task that employs contextualized representations.

3.3.1 Background

Medical errors are a pervasive problem in healthcare that can result in serious patient harm [121]. To identify and reduce the occurrence of preventable errors, many medical centers use reporting systems to document cases. Initial reports are often reviewed and revised by more experienced physicians. The revisions could be due to stylistic reasons or (more importantly) misinterpretations/errors in the initial report. In such cases, to prevent recurrence of the errors, is crucial to identify reports with substantive differences between the original and final report and discuss them with the clinician who wrote the initial report. It is often challenging to manually identify such cases among the large number of daily written reports in a timely manner. In this work, we propose an approach for ranking revisions of medical reports by the degree of discrepancy between the different versions of the report. This allows medical practitioners to easily find the reports in which they made an error, which helps them learn from their mistakes and prevent future similar errors.

This is a challenging task to automate because the edits that an attending physician makes to a report can range from stylistic differences to significant discrepancies that may have a major effect on the patient (e.g., an unobserved mass). See Figure 3.2 for an example of significant and non-significant discrepancies from radiology reports. As we can see, differences between the significant and non-significant discrepancies are often not trivial to identify and requires more than just comparing surface word changes in the versions of the reports. Furthermore, significant discrepancies can occur relatively frequently in practice; in our dataset collected from a large urban hospital, around 7% of reports contained significant errors. With hundreds of reports generated a week at some hospitals, this can amount to a considerable number of errors. We address this problem by proposing a supervised ranking approach for clinician's

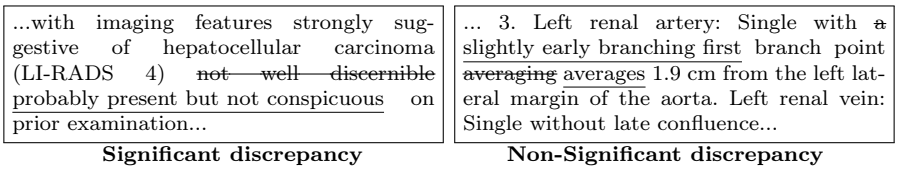

Figure 3.2 Example radiology impression revisions. Strikeout indicates removed content and underline indicates added content. We aim to rank report revisions by the significance of the discrepancy.

revised reports by the degree that there are significant discrepancies between their preliminary report and the final corrected report. I.e., our goal is to rank revisions that are more likely due to errors higher than revisions that are merely due to stylistic changes.

Prior works have investigated significant discrepancies in medical reports through comparison of surface textual features [171, 205], semantic similarity features [25], and word frequencies [82]. These works often treat the problem as classification and the most successful ones leverage a variety of textual similarity measures. Viewing this problem as ranking is a more suitable and practical form of evaluation; given a doctor's limited time, it is important for them to be presented with the reports that have the most significant discrepancies.

Document ranking in the broad medical domain have received extensive interest of researchers [87, 165, 174, 178, 189, 225]. However, these efforts focus on conventional query-document retrieval. Our goal is to rank significant discrepancies by measuring the semantic overlap between the initial and final report. There have also been efforts to identify semantic similarity between two texts, e.g., for paraphrase identification [54, 102, 162, 196], but these approaches operate on the sentence-level, making them unsuitable for documents (e.g., radiology reports).

3.3.2 Methodology

We propose a supervised model that measures the overlap between the preliminary and final report for the purpose of ranking pairs of preliminary and final reports based on their significance over a given period of time. We observe that the central challenge of this task is being permissive of surface-level changes (which may be considerable), while emphasizing changes of substance, which may be subtle (see Figure 3.2 for examples of such changes). To address this, we incorporate *importance* and *similarity* scores. The *importance score* weights each term/phrase and is learned during training. This score allows for terms that are not important to have less of an impact on the ranking score of the report (e.g., words like *well* and *but*). Note that this is a special application in which some function words that are often ignored actually have a big impact on the meaning of a report (e.g., *not* is often considered a stop word and removed). We let the model learn which terms are important during training. The *matching score* allows for the model to account for the replacement of similar terms using the cosine distance of word vectors (e.g., *averaging* and *averages* are similar) and synonym information from a domain-specific ontology (*chauffeur fracture* and *Hutchinson fracture* are synonymous). This allows the replacement of semantically-similar terms to have little impact on the ranking score. We calculate three *similarity scores* (addition, deletion, and overlap) using the importance and matching scores, and linearly combine them as a *ranking score*.

Notation and task definition. Let R be a set of clinical reports. Each report $r \in R$ consists of a preliminary and final version of the report (p and f, respectively), and a label $l \in \{0, 1, ..., L\}$ indicating the degree of discrepancy between p and f. Each version of the report consists of a sequence of tokens, denoted by p_i and f_i. The significant discrepancy ranking task produces a ranking score $s \in \mathbb{R}$ for each report

$r \in R$ such that the reports with higher degrees of discrepancy are assigned a higher ranking score.

Similarity scores. Our approach combines several similarity scores to produce a ranking score. Specifically, we measure the weighted soft additions, deletions, and overlap of unigrams, n-grams, and ontological entities. The addition score (S_a, Eq. 3.2) defines weighted soft similarity as the ratio between the similarity score (weighted by a learned importance score) and the total importance of all terms in the final report. Thus, terms from the final report that do not appear in the preliminary report (i.e., additions) yield a higher score. The deletion score (S_d, Eq. 3.3) is defined similarly, but in terms of the preliminary report; terms from the preliminary report that do not appear in the final report (deletions) yield a higher score. The overlap score (S_o, Eq. 3.4) combines the addition and deletion scores into one succinct measure. We use all three scores to measure term unigram, n-gram, and ontological differences (defined below). We define the similarity functions (where $M_X(y) \in [0, 1]$ is a matching score of term y in X, and $I(y) \in [0, 1]$ is the importance score of term y) as:

$$S_a(p, f) = -\frac{\sum_{f_i \in f} M_p(f_i) I(f_i)}{\sum_{f_i \in f} I(f_i)} \tag{3.2}$$

$$S_d(p, f) = -\frac{\sum_{p_i \in p} M_f(p_i) I(p_i)}{\sum_{p_i \in p} I(p_i)} \tag{3.3}$$

$$S_o(p, f) = -\frac{\sum_{p_i \in p} M_f(p_i) I(p_i) + \sum_{f_i \in f} M_p(f_i) I(f_i)}{\sum_{p_i \in p} I(p_i) + \sum_{f_i \in f} I(f_i)} \tag{3.4}$$

Unigram and n-gram matching. Unigram matching can provide valuable signals for significance in radiology reports. For instance, the addition *no* (e.g., *fracture* vs. *no fracture*) could change the meaning of the report considerably. We define the matching function for unigrams as the maximum cosine similarity between the word embeddings ($emb(\cdot)$) of the term and any term in the other report, and a unigram

importance function using a simple feed-forward layer with sigmoid activation (W_{imp} and b_{imp} as model parameters):

$$M_X(y) = \max_{x \in X}(\cos(emb(x), emb(y))) \tag{3.5}$$

$$I(y) = \sigma(emb(y)W_{imp} + b_{imp}) \tag{3.6}$$

N-gram matching provides another important view of similarity, since there are many multi-word noun phrases in radiological notes. For instance, *right arm* and *left arm* represent completely different parts of the body, and should be treated differently. We handle n-grams by first taking the average of the embeddings over sliding windows. This is a simple and effective way to combine the representations. We use bi-grams and tri-grams in our experiments.

Ontological matching. Since medical knowledge is broad and extensive, the model may never encounter certain medical entities during training. This knowledge may also not be captured effectively by embeddings. Thus, it is valuable to explicitly encode domain information into the model using an ontology. We use a mapping function that matches any exact ontological name to the corresponding concept, a constant similarity for exact entity matches, and constant weight for all ontology concepts. We use RadLex (v4.0, `http://radlex.org/`), an ontology of radiology concepts (e.g., procedures, diagnoses, etc.).

3.3.3 Experiment

Dataset. We train and evaluate using a dataset of 3,368 radiology reports from a large urban hospital. Each sample consists of a preliminary report written by a resident, and a final report revised by the attending radiologist, who labeled the edit by the degree of discrepancy between the two reports. The labels are `0` (attending doctor fully agrees with assessment of the resident, 81% of reports), `1` (errors exist, but they

are insignificant to the overall impression, 12%), 2 (subtle, yet important, error exists, 6%), 3 (an obvious error exists, 1%). We split the dataset into 122 sets based on the combination of resident and week (*ranking sets*, average 27.6 reports per ranking set, min 5, max 148). Since residents often work weekly shifts, this is a valuable setting because it allows residents to review report discrepancies from the past week. We randomly split the ranking sets into 60-20-20 train-dev-test set splits. Each ranking set consists of at least 5 reports, each with at least one report discrepancy. Radiology reports contain several sections; we primarily concern ourselves with the summary section of the reports (called the *impression*) because it contains the main findings.

Baselines. To evaluate the effectiveness of the model, we compare with the variety of methods in the state-of-the-art, including ranking models, domain specific models and textual similarity models, briefly described below:

- **Vector space model (VSM)**. We use the traditional TFIDF-weighted vector space similarity score between the preliminary and final report (from lucene).
- **BiPACRR**. We test the PACRR [75] neural IR model because it learns to identify n-gram similarity between two texts. We modify the architecture to learn two scores (one with the preliminary report as the query and the other with the final report as the query), and linearly combine them to produce a final ranking score. We call this variant BiPACRR. We also experimented with other neural rankers (e.g., KNRM [214]), but BiPACRR was the most effective.
- **Textual similarity regression (SimReg)** [25]. This approach uses logistic regression to combine several hand-crafted features (mostly consisting of textual similarity measures and lexical features) to identify significant discrepancies in radiology reports. Since this approach performs classification, we use the label score as the ranking score. Our experiments used the authors' implementation.

- **(Sci)BERT classification**. We use the standard fine-tuned BERT textual similarity method on both the pretrained BERT [44] (`base-uncased`) and SciBERT [10] (`scivocab-uncased`) models. Based on preliminary parameter tuning, we use a learning rate of 10^{-5} for fine-tuning these models.

Evaluation metrics. Given the time constraints of doctors, we choose evaluation metrics that emphasize placing reports with higher discrepancies at the top. We evaluate using nDCG@1, nDCG@5, nDCG (without cutoff), P@1, P@5, and R-Prec (binary labels test any degree of discrepancy higher than label `0`).

Parameters and training. We train the neural models using pairwise cross-entropy loss [43]. Hyper-parameters are tuned using nCDG@5 on the dev set. We use SciBERT term embeddings [10] in our model and BiPACRR and tune for the optimal layer's embeddings akin to [110]. SciBERT is an adaptation of BERT to the biomedical and scientific domains, making it suitable for radiology notes.

Results. Test set performance of our best model configuration are shown in Table 3.3. Our optimal model consists of unigram, bi-gram, tri-gram, and RadLex scores. When compared to the best prior work (SimReg [25]), our model typically yields a considerable improvement in ranking performance. Our method improves R-Prec by 7.4%, nDCG@1 and nDCG@5 performance by 4.5%, and P@5 performance by 4.7%. In 54% of the test cases, our approach improves the nDCG@5 score over SimReg (decreases performance in only 27% of cases). Our model also outperforms leading language model classification approaches (BERT and SciBERT) and a leading neural ranking approach tuned for this task in most metrics (BiPACRR) by up to 15.4% in nDCG@1. We attribute this improved effectiveness of our approach to the explicit modeling of term importance and overlap, which are critical for the task.

Ablations. Table 3.4 shows the ablation study examining the importance of different components in our system. We observe that both contextualization and domain-

Table 3.3 Radiology ranking performance of our method and baselines.

Model	nDCG@1	nDCG@5	nDCG	P@1	P@5	R-Prec
VSM	48.1	54.0	70.9	65.4	42.3	49.4
BERT	59.0	69.8	78.7	69.2	53.8	53.9
SciBERT	62.2	68.2	79.3	76.9	51.5	58.3
BiPACRR	64.1	68.6	77.5	69.2	**56.2**	55.3
SimReg	69.9	70.7	81.1	**80.8**	51.5	51.8
Our Method	**74.4**	**75.2**	**83.7**	**80.8**	**56.2**	**59.2**

Table 3.4 Ablation study of our radiology ranking method.

Model	nDCG@5
Full Model	**75.2**
- replace SciBERT with BERT	64.7
- replace SciBERT with BioNLP (`pubmed-pmc`, `bio.nlplab.org`)	62.2
- replace SciBERT with FastText (`wiki-news-300d-1M`, `fasttext.cc`)	59.1
- without term importance	68.3
- without ontology similarity	65.4
- only overlap score (S_o)	70.8
- only addition/deletion scores (S_a and S_d)	61.5

specificity of the word embeddings improve the performance of our approach. The term importance mechanism improves nDCG@5 by 6.9% and the ontology similarity improves performance by 9.8%. All three similarity measures appear to be important, however the overlap score alone can account for most of the performance (last row in table). This may be because it succinctly accounts for both additions and deletions.

Term importance. To better understand the term importance mechanism of our approach, we present an example report in Figure 3.3 (slightly altered for privacy). This report contains highly significant discrepancies and was ranked at position 3 by

~~anteroinferior~~ ~~dislocation~~ ~~of~~ ~~the~~ ~~left~~ ~~shoulder.~~ ~~mild~~ ~~hill-~~ ~~sachs~~ ~~deformity~~ ~~without~~ ~~associated~~ ~~bankart~~ ~~lesion.~~ no evidence of acute fracture or dislocation ~~of~~ ~~the~~ ~~humerus~~.

Figure 3.3 Example unigram importance scores from our radiology model. Results are the mean of the scores from the preliminary and final report. Darker colors indicates higher importance scores. Underlines indicate additions to the final report; strikeouts indicate deletions.

our approach and position 9 by SimReg (below several non-significant discrepancies). We observe that our model considers many radiological conditions as important, both when unmodified between the reports and when added/deleted (e.g., *fracture*, *dislocation*, *bankart*). Judging by the low textual similarity in this example, we conclude that the SimReg model may be relying too heavily on lexical features. We check the terms that are assigned high importance scores across all reports and find the most common are *no* (12% of reports), *cardiopulmonary* (3%), *process* (3%), and *abnormality* (3%).

3.3.4 Summary

We presented a supervised ranking model based on lexical and ontological overlaps to rank medical reports by their discrepancy significance. On a real-world dataset of medical reports, we demonstrated that our approach outperforms existing approaches by large margins. This direction is a critical step towards addressing the problem of medical errors. By allowing medical practitioners to more easily find and learn from their previous errors, the chance of recurrent errors will be reduced, improving the well-being of patients.

3.4 Addressing the Lack of Multi-lingual Training Data

We recognize the fact that the majority of ad-hoc retrieval research is done using English data. This is largely due to the fact that most training and evaluation resources are in English. Here, we study how effective neural ranking methods are when trained on abundant English data, and evaluated on data in other languages. Furthermore, we evaluated the effect of introducing a small amount of in-language training data.

3.4.1 Background

Every day, billions of non-English speaking users [169] interact with search engines; however, commercial retrieval systems have been traditionally tailored to English queries, causing an information access divide between those who can and those who cannot speak this language [227]. Non-English search applications have been equally under-studied by most information retrieval researchers. Historically, ad-hoc retrieval systems have been primarily designed, trained, and evaluated on English corpora (e.g., [4, 20, 21, 125]). More recently, a new wave of supervised state-of-the-art ranking models have been proposed by researchers [56, 75, 128, 141, 214, 220]; these models rely on neural architectures to rerank the head of search results retrieved using a traditional unsupervised ranking algorithm, such as BM25. Like previous ad-hoc ranking algorithms, these methods are almost exclusively trained and evaluated on English queries and documents.

The absence of rankers designed to operate on languages other than English can largely be attributed to a lack of suitable publicly available data sets. This aspect particularly limits supervised ranking methods, as they require samples for training and validation. For English, previous research relied on English collections such as

TREC Robust 2004 [202], the 2009-2014 TREC Web Track [27], and MS MARCO [9]. No datasets of similar size exist for other languages.

While most of recent approaches have focused on ad hoc retrieval for English, some researchers have studied the problem of cross-lingual information retrieval. Under this setting, document collections are typically in English, while queries get translated to several languages; sometimes, the opposite setup is used. Throughout the years, several cross lingual tracks were included as part of TREC. TREC 6, 7, 8 [16] offer queries in English, German, Dutch, Spanish, French, and Italian. For all three years, the document collection was kept in English. CLEF also hosted multiple cross-lingual ad-hoc retrieval tasks from 2000 to 2009 [15]. Early systems for these tasks leveraged dictionary and statistical translation approaches, as well as other indexing optimizations [152]. More recently, approaches that rely on cross-lingual semantic representations (such as multilingual word embeddings) have been explored. For example, Vulic and Moens [204] proposed BWESG, an algorithm to learn word embeddings on aligned documents that can be used to calculate document-query similarity. Sasaki et al [179] leveraged a data set of Wikipedia pages in 25 languages to train a learning to rank algorithm for Japanese-English and Swahili-English cross-language retrieval. Litschko et al [101] proposed an unsupervised framework that relies on aligned word embeddings. Ultimately, while related, these approaches are only beneficial to users who can understand documents in two or more languages instead of directly tackling non-English document retrieval.

A few monolingual ad-hoc data sets exist, but most are too small to train a supervised ranking method. For example, TREC produced several non-English test collections: Spanish [60], Chinese Mandarin [199], and Arabic [140]. Other languages were explored, but the document collections are no longer available. The CLEF initiative includes some non-English monolingual datasets, though these are primarily focused

on European languages [15]. Recently, Zheng et al. [229] introduced Sogou-QCL, a large query log dataset in Mandarin. Such datasets are only available for languages that already have large, established search engines.

Inspired by the success of neural retrieval methods, this work focuses on studying the problem of monolingual ad-hoc retrieval on non English languages using supervised neural approaches. In particular, to circumvent the lack of training data, we leverage transfer learning techniques to train Arabic, Mandarin, and Spanish retrieval models using English training data. In the past few years, transfer learning between languages has been proven to be a remarkably effective approach for low-resource multilingual tasks (e.g. [80, 85, 181, 221]). Our model leverages a pre-trained multilanguage transformer model to obtain an encoding for queries and documents in different languages; at train time, this encoding is used to predict relevance of query document pairs in English. We evaluate our models in a zero-shot setting; that is, we use them to predict relevance scores for query document pairs in languages never seen during training. By leveraging a pre-trained multilingual language model, which can be easily trained from abundant aligned [89] or unaligned [29] web text, we achieve competitive retrieval performance without having to rely on language specific relevance judgements.

3.4.2 Methodology

Zero-shot Multi-Lingual Ranking. Because large-scale relevance judgments are largely absent in languages other than English, we propose a new setting to evaluate learning-to-rank approaches: zero-shot cross-lingual ranking. This setting makes use of relevance data from one language that has a considerable amount of training data (e.g., English) for model training and validation, and applies the trained model to a different language for testing.

More formally, let $\mathcal{S}$ be a collection of relevance tuples in the source language, and $\mathcal{T}$ be a collection of relevance judgments from another language. Each relevance tuple $\langle \mathbf{q}, \mathbf{d}, r \rangle$ consists of a query, document, and relevance score, respectively. In typical evaluation environments, $\mathcal{S}$ is segmented into multiple splits for training ($\mathcal{S}_{train}$) and testing ($\mathcal{S}_{test}$), such that there is no overlap of queries between the two splits. A ranking algorithm is tuned on $\mathcal{S}_{train}$ to define the ranking function $R_{\mathcal{S}_{train}}(\mathbf{q}, \mathbf{d}) \in \mathbb{R}$, which is subsequently tested on $\mathcal{S}_{test}$. We propose instead tuning a model on all data from the source language (i.e., training $R_{\mathcal{S}}(\cdot)$), and testing on a collection from the second language ($\mathcal{T}$).

3.4.3 Experiment

Experimental Setup

Datasets. We evaluate on monolingual newswire datasets from three languages: Arabic, Mandarin, and Spanish. The Arabic document collection contains $384k$ documents (LDC2001T55), and we use topics/relevance information from the 2001–02 TREC Multilingual track (25 and 50 topics, respectively). For Mandarin, we use $130k$ news articles from LDC2000T52. Mandarin topics and relevance judgments are utilized from TREC 5 and 6 (26 and 28 topics, respectively). Finally, the Spanish collection contains $58k$ articles from LDC2000T51, and we use topics from TREC 3 and 4 (25 topics each). We use the topics, rather than the query descriptions, in all cases except TREC Spanish 4, in which only descriptions are provided. The topics more closely resemble real user queries than descriptions.[8] We test on these collections because they are the only document collections available from TREC at this time.[9]

[8]Some have observed that the context provided by query descriptions are valuable for neural ranking, particularly when using contextualized language models [36].

[9]`https://trec.nist.gov/data/docs_noneng.html`

We index the text content of each document using a modified version of Anserini with support for the languages we investigate [217]. Specifically, we add Anserini support for Lucene's Arabic and Spanish light stemming and stop word list (via `SpanishAnalyzer` and `ArabicAnalyzer`). We treat each character in Mandarin text as a single token.

Modeling. We explore the following ranking models:

- **Unsupervised baselines**. We use the Anserini [217] implementation of BM25, RM3 query expansion, and the Sequential Dependency Model (SDM) as unsupervised baselines. In the spirit of the zero-shot setting, we use the default parameters from Anserini (i.e., assuming no data of the target language).

- **PACRR** [75] models n-gram relationships in the text using learned 2D convolutions and max pooling atop a query-document similarity matrix.

- **KNRM** [214] uses learned Gaussian kernel pooling functions over the query-document similarity matrix to rank documents.

- **Vanilla BERT**, as introduced in Section 3.2, uses the BERT [44] transformer model, with a dense layer atop the classification token to compute a ranking score. To support multiple languages, we use the `base-multilingual-cased` pretrained weights. These weights were trained on Wikipedia text from 104 languages.

We use the embedding layer output from `base-multilingual-cased` model for PACRR and KNRM. In pilot studies, we investigated using cross-lingual MUSE vectors [29] and the output representations from BERT, but found the BERT embeddings to be more effective.

Experimental Setup. We train and validate models using TREC Robust 2004 collection [202]. TREC Robust 2004 contains 249 topics, $528k$ documents, and $311k$

relevance judgments in English (folds 1-4 from [77] for training, fold 5 for validation). Thus, the model is only exposed to English text in the training and validation stages (though the embedding and contextualized language models *are* trained on large amounts of unlabeled text in the languages). The validation dataset is used for parameter tuning and for the selection of the optimal training epoch (via nDCG@20). We train using pairwise softmax loss with Adam [86].

We evaluate the performance of the trained models by re-ranking the top 100 documents retrieved with BM25. We report MAP, Precision@20, and nDCG@20 to gauge the overall performance of our approach, and the percentage of judged documents in the top 20 ranked documents (judged@20) to evaluate how suitable the datasets are to approaches that did not contribute to the original judgments.

Results

We present the ranking results in Tables 3.5 and 3.6. We first point out that there is considerable variability in the performance of the unsupervised baselines; in some cases, RM3 and SDM outperform BM25, whereas in other cases they under-perform. Similarly, the PACRR and KNRM neural models also vary in effectiveness, though more frequently perform much worse than BM25. This makes sense because these models capture matching characteristics that are specific to English. For instance, n-gram patterns captured by PACRR for English do not necessarily transfer well to languages with different constituent order, such as Arabic (VSO instead of SVO). An interesting observation is that the Vanilla BERT model (which recall is only tuned on English text) generally outperforms a variety of approaches across three test languages. This is particularly remarkable because it is a single trained model that is effective across all three languages, without any difference in parameters. The exceptions are the Arabic 2001 dataset, in which it performs only comparably to BM25

Table 3.5 Zero-shot multi-lingual results for various baseline and neural methods in Arabic and Mandarin.

Ranker	P@20	nDCG@20	MAP	judged@20
Arabic (TREC 2002) [140]				
BM25	0.3470	0.3863	0.2804	99.0%
BM25 + RM3	0.3320	0.3705	↓ 0.2641	95.1%
SDM	0.3380	0.3775	↓ 0.2572	98.1%
PACRR multilingual	0.3270	0.3499	↓ 0.2517	96.4%
KNRM multilingual	0.3210	↓ 0.3415	↓ 0.2503	95.2%
Vanilla BERT multilingual	↑ **0.3790**	**0.4205**	**0.2876**	97.4%
Arabic (TREC 2001) [140]				
BM25	**0.5420**	**0.5933**	**0.3462**	97.2%
BM25 + RM3	↓ 0.4700	0.5458	↓ 0.2903	85.6%
SDM	0.5140	0.5843	0.3213	96.2%
PACRR multilingual	↓ 0.3880	↓ 0.3933	↓ 0.2724	90.6%
KNRM multilingual	↓ 0.4140	↓ 0.4327	↓ 0.2742	91.0%
Vanilla BERT multilingual	0.5240	0.5628	0.3432	91.0%
Mandarin (TREC 6) [199]				
BM25	0.5962	0.6409	0.3316	89.6%
BM25 + RM3	↓ 0.5019	↓ 0.5571	0.2696	75.6%
SDM	0.5942	0.6320	0.3472	92.1%
PACRR multilingual	↓ 0.4923	↓ 0.5238	0.2856	79.0%
KNRM multilingual	↓ 0.5308	↓ 0.5497	↓ 0.3107	80.8%
Vanilla BERT multilingual	↑ **0.6615**	↑ **0.6959**	↑ **0.3589**	92.7%
Mandarin (TREC 5) [203]				
BM25	0.3893	0.4113	0.2548	85.4%
BM25 + RM3	↓ 0.2768	↓ 0.3021	↓ 0.1698	64.6%
SDM	↑ 0.4536	↑ 0.4744	↑ 0.2855	94.1%
PACRR multilingual	0.3786	0.3998	0.2331	83.2%
KNRM multilingual	↓ 0.3232	↓ 0.3449	↓ 0.2223	77.5%
Vanilla BERT multilingual	↑ **0.4589**	↑ **0.5196**	↑ **0.2906**	92.0%

Significant improvements and reductions in performance compared with BM25 are indicated with ↑ and ↓, respectively (paired t-test by query, $p < 0.05$).

Table 3.6 Zero-shot multi-lingual results for various baseline and neural methods in Spanish.

Ranker	P@20	nDCG@20	MAP	judged@20
Spanish (TREC 4) [60]				
BM25	0.3080	0.3314	0.1459	83.8%
BM25 + RM3	0.3360	0.3358	↑ **0.2024**	85.2%
SDM	0.2780	0.3061	0.1377	78.6%
PACRR multilingual	0.2440	0.2494	0.1294	69.4%
KNRM multilingual	0.3120	0.3402	0.1444	79.2%
Vanilla BERT multilingual	↑ **0.4400**	↑ **0.4898**	↑ 0.1800	85.6%
Spanish (TREC 3) [61]				
BM25	0.5220	0.5536	0.2420	84.8%
BM25 + RM3	↑ 0.6100	0.6236	↑ **0.3887**	93.0%
SDM	0.4920	0.5178	0.2258	83.8%
PACRR multilingual	↓ 0.4140	↓ 0.4092	0.2260	76.0%
KNRM multilingual	0.5560	0.5700	0.2449	85.2%
Vanilla BERT multilingual	↑ **0.6400**	↑ **0.6672**	↑ 0.2623	90.8%

Significant improvements and reductions in performance compared with BM25 are indicated with ↑ and ↓, respectively (paired t-test by query, $p < 0.05$).

Table 3.7 Zero-Shot (ZS) and Few-Shot (FS) comparison for Vanilla BERT (multilingual) on each dataset.

	P@20		nDCG@20		MAP	
Dataset	ZS	FS	ZS	FS	ZS	FS
Arabic 2002	**0.3790**	0.3690	**0.4205**	0.3905	**0.2876**	0.2822
Arabic 2001	0.5240	↑ **0.6020**	0.5628	↑ **0.6405**	0.3432	**0.3529**
Mandarin 6	0.6615	**0.6808**	0.6959	**0.7099**	**0.3589**	0.3537
Mandarin 5	0.4589	**0.4643**	**0.5196**	0.5014	**0.2906**	0.2895
Spanish 4	0.4400	↑ **0.5060**	0.4898	↑ **0.5636**	0.1800	↑ **0.2020**
Spanish 3	0.6400	**0.6560**	0.6672	**0.6825**	0.2623	**0.2684**

Within each metric and dataset, the top result is listed in bold. Significant increases from using FS are indicated with ↑ (paired t-test, $p < 0.05$).

and the MAP results for Spanish. For Spanish, RM3 is able to substantially improve recall (as evidenced by MAP), and since Vanilla BERT acts as a re-ranker atop BM25, it is unable to take advantage of this improved recall, despite significantly improving the precision-focused metrics. In all cases, Vanilla BERT exhibits judged@20 above 85%, indicating that these test collections are still valuable for evaluation.

To test whether a small amount of in-language training data can further improve BERT ranking performance, we conduct an experiment that uses the other collection for each language as additional training data. The in-language samples are interleaved into the English training samples. Results for this few-shot setting are shown in Table 3.7. We find that the added topics for Arabic 2001 (+50) and Spanish 4 (+25) significantly improve the performance. This results in a model significantly better than BM25 for Arabic 2001, which suggests that there may be substantial distributional differences in the English TREC 2004 training and Arabic 2001 test collections. We further back this up by training an "oracle" BERT model (training on the

test data) for Arabic 2001, which yields a model substantially better (P@20=0.7340, nDCG@20=0.8093, MAP=0.4250).

3.4.4 Summary

We introduced a zero-shot multilingual setting for evaluation of neural ranking methods. This is an important setting due to the lack of training data available in many languages. We found that contextualized languages models (namely, BERT) have a big upper-hand, and are generally more suitable for cross-lingual performance than prior models (which may rely more heavily on phenomena exclusive to English). We also found that additional in-language training data may improve the performance, though not necessarily.

3.5 Choosing Good Training Samples

In Section 2.3, we demonstrated that dataset statistics can be used Incorporated as features and signals in non-contextualized neural rankers. In this section, we explore whether dataset statistics can be applied to contextualized models during the training process, rather than as model features. This is formulated as a *Curriculum Learning* problem, i.e., focusing on approaches for selecting better training samples.

We motivate our approach with the simple intuition that some answers are easier to assess the relevance of than others. For instance, consider a question about the health impacts of vegetarianism (see Figure 3.4). A passage written explicitly about this topic (e.g., (a)) should be relatively straightforward to identify, as it uses many of the terms in the question. This likely yields a high ranking position using conventional probabilistic approaches, such as BM25. A passage written about the health benefits of *veganism* (a more strict version of vegetarianism) may also answer the question (b).

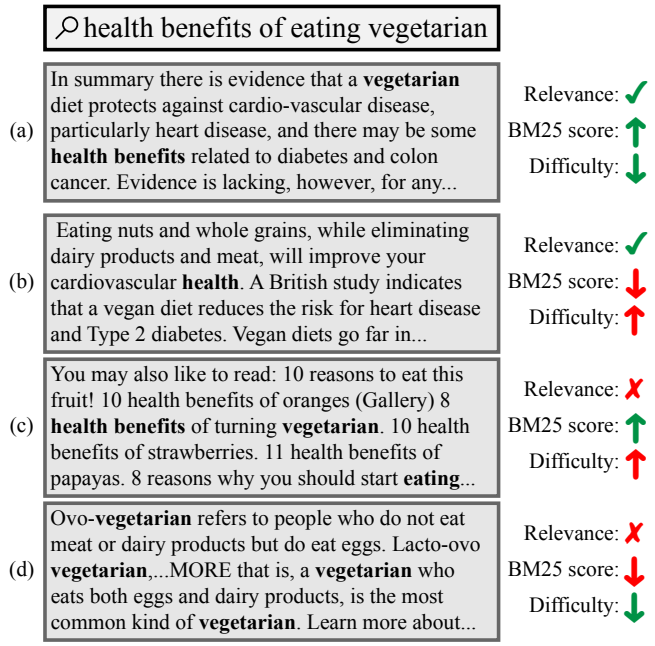

Figure 3.4 Example of curriculum approach from MS-MARCO dataset. In this example, we predict (a) is 'easy' because it is relevant and has a high BM25 score. (d) is likewise 'easy' weight because it is non-relevant and has a low score. (b) is a 'difficult' sample because is relevant, yet has a low score due to the few term matches. We also predict (c) to be 'difficult' because it is non-relevant, yet it has a high score. Our approach begins by weighting 'easy' training samples high and 'difficult' training samples low.

However, it involves more complicated reasoning and inference (such as the understanding of the relationship between the two diets) and semantic understanding of the way in which the content is presented. Similarly, consider two non-relevant answers: one that matches most of the query terms (c) and one that does not (d). We argue that the former is more difficult for the ranking model to identify as non-relevant due to the large number of matching terms, and the latter is easier due to critical missing terms (e.g., health benefits).

While an ideal ranker would rank both (a) and (b) high, doing so we may add noise and complexity to the model that reduces the overall quality of ranking. Specifically, ranking (b) high may make it more difficult to identify (c) and (d) as non-relevant. Our method attempts to overcome this issue by forcing the ranker to focus primarily on the "easy" training samples before gradually moving on to learning to rank all training samples via training sample weighting.

We formulate this idea using the *curriculum learning* (CL) framework [11]. Learning through a curriculum is an idea borrowed from cognitive sciences according to which the learning process follows a multi-step training path. Initially, the learning algorithm is trained by using simple examples and smooth loss functions. Then it is progressively fine-tuned so as to deal with examples and loss functions of increasing complexity. We instantiate the CL framework in the learning to rank domain by assigning different weights to training samples through a heuristic. In early stages of training, high weights are assigned to *easy* training pairs while *difficult* samples are given low weights. As training progresses, we gradually smooth this imbalance. Eventually, all training samples are weighted equally, regardless of the estimated difficulty.

To estimate the difficulty of question-answer pairs and to choose the training weight accordingly, we consider both information from an unsupervised baseline ranker (e.g., BM25) and the human-assessed relevance of the answer to the given question (see Figure 3.4). When an unsupervised ranker is able to identify the example effectively (i.e., it ranks a relevant document high or a non-relevant document low) the training sample is considered as "easy". On the other hand, when the unsupervised ranker fails to correctly score them, the sample is considered as "difficult".

We show that our approach can be easily integrated into a neural ranking pipeline. We validate our approach using three weighting heuristics based on an unsupervised

ranker using two leading neural ranking methods (BERT [44] and ConvKNRM [37]). Our results show significant ranking improvements when tested on three open-domain (i.e., not domain-specific) answer ranking benchmarks: TREC Deep Learning (DL), TREC Complex Answer Retrieval (CAR), and ANTIQUE. These datasets vary in scale (hundreds of thousands of answers to tens of millions) and source of relevance information (graded or positive-only, human-annotated or inferred from document structure). We test using both pointwise and pairwise losses.

3.5.1 Background

Curriculum Learning (CL) can be considered a particular case of *Continuation Methods*, generally used when the target objective function is non-convex and its direct optimization may lead the training to converge to a poor local minimum [11, 26]. The basic idea to overcome this problem through a curriculum approach is to organize the learning process as a path where the easiest training samples are presented first and the complexity of the following ones is gradually increased. This strategy allows the learner to exploit previously seen concepts to ease the acquisition of subsequent more difficult ones. CL approaches are proved successful for training neural networks in different domains such as NLP [28, 72], language models (not used for ranking tasks) [11], image representation [22], network representation [157]. To our knowledge, the only attempt to explore how CL methods can be exploited in the document ranking domain is the one by Ferro *et al.* [53] where authors exploit the curriculum learning strategy in a gradient boosting algorithm that learns ranking models based on ensembles of decision trees. The results reported by Ferro *et al.* show that a curriculum learning strategy gives only a limited boost to the ranking performance of an ensemble of decision trees. Similar to our approach, Fidelity-weighted learning [42] applies weights to training samples for learning ranking models. However, this

approach focuses on estimating the quality of weak labels (namely, treating BM25 scores as labels), rather than the difficulty of training samples with higher-quality labels (e.g., human-annotated labels).

Sachan and Xing [172] propose curriculum learning approaches for question answering, but in a closed-domain setting. In open-domain question answering, there are several challenges encountered, including that there is a much larger collection of answers to draw from (millions of answers) and multiple correct answers to a given question. Thus, we tackle this problem from an IR-perspective, utilizing signals from ranking models. Recently, Penha and Hauff [150] propose approaches for using curriculum learning to rank conversational responses, yielding up to a 2% improvement in ranking effectiveness. The curricula proposed are specific to the domain of conversational responses and are non-trivial to apply to other domains. In contrast, we propose simple heuristics based on initial retrieval ranks and scores, and we show their effectiveness across multiple ranking models, loss functions, and answer ranking datasets in an open-domain setting.

3.5.2 Methodology

We present our approach for applying curriculum learning to the training of neural rankers. At a high level, our approach applies a heuristic to determine the difficulty of a particular training sample. This difficulty estimation is then used for weighting training samples. In early stages of training, samples that the heuristic predicts as easy are given a higher weight, while samples predicted to be difficult are given a lower weight. Gradually, our approach eases off this crutch (controlled by a new hyper-parameter). Eventually, all training samples are weighted equally, regardless of the estimated difficulty.

Table 3.8 Table of symbols for curriculum learning.

Symbol	Definition
R_θ	Neural ranking function with parameters θ
$\mathcal{L}$	Loss function
$\mathcal{D}$	Training sample difficulty function
W	Training sample weight function
$\mathbf{q}$	Query (i.e., question)
$\mathbf{d}$	Document (i.e., answer)
$\mathbf{d}^+$	Relevant document
$\mathbf{d}^-$	Non-relevant document
$\mathbf{D}$	Set of ranked documents
s	Manual relevance assessment score
T	Collection of training data
t	Training sample from T
i	Training iteration (epoch)
m	End of curriculum iteration (hyperparameter)

Our approach allows for fair comparisons to an unmodified training process because no changes are made to the selection of the training data itself; the effect is only on the weight of the sample during training. Furthermore, this allows for an easy integration of our approach into existing training pipelines; no changes to the data selection technique are required, and the heuristics rely on information readily available in most re-ranking settings.

Our approach degrades into the typical training process in two ways: either (1) a heuristic can be used that gives every sample a weight of 1, or (2) the hyper-parameter that drives the degradation of the approach to equal weighting can be set to immediately use equal weights.

Notation and Preliminaries

A summary of the symbols used is given in Table 3.8. Let an ad-hoc neural ranking model be represented as $R_\theta(\mathbf{q}, \mathbf{d}) \in \mathbb{R}$, which maps a given query-document pair $(\mathbf{q}, \mathbf{d})$ to a real-valued relevance score given the model parameters θ. For simplicity, we refer to questions as queries and answers as documents. Through a set of training data points $t \in T$ and a loss function $\mathcal{L}(t)$, the model parameters θ are optimized to maximize the ranking performance. The training data sample $t \in T$ depends on the type of loss employed. Two common techniques employed for training neural rankers rely on pointwise or pairwise loss. For pointwise loss, training data consists of triples $t_{point} = \langle \mathbf{q}, \mathbf{d}, s \rangle$, where $\mathbf{q}$ is a query, $\mathbf{d}$ is a document, and s is its relevance score, e.g., the relevance score given to the query-document pair by a human assessor. The loss for this sample often uses squared error between the predicted score and the relevance score s:

$$\mathcal{L}^{point}(\mathbf{q}, \mathbf{d}, s) = \big(s - R_\theta(\mathbf{q}, \mathbf{d})\big)^2 \tag{3.7}$$

On the other hand, pairwise loss uses two document samples for the same query (one relevant and one non-relevant), and optimizes to assign a higher score to the relevant document than the non-relevant one. Training triples for pairwise loss are represented as $t_{pair} = \langle \mathbf{q}, \mathbf{d}^+, \mathbf{d}^- \rangle$, where $\mathbf{q}$ is the query, $\mathbf{d}^+$ is the relevant document, and $\mathbf{d}^-$ is the non-relevant document. One common pairwise loss function is the softmax cross-entropy loss:

$$\mathcal{L}^{pair}(\mathbf{q}, \mathbf{d}^+, \mathbf{d}^-) = \frac{\exp\big(R_\theta(\mathbf{q}, \mathbf{d}^+)\big)}{\exp\big(R_\theta(\mathbf{q}, \mathbf{d}^+)\big) + \exp\big(R_\theta(\mathbf{q}, \mathbf{d}^-)\big)} \tag{3.8}$$

Curriculum Framework for Answer Ranking

Let a difficulty function $\mathcal{D} : T \mapsto [0, 1]$ define a weight $\mathcal{D}(t)$ for the training sample $t \in T$. Without loss of generality we now assume that a high value of $\mathcal{D}(t)$, i.e.,

a value close to 1, represents an easy sample, while a low value, i.e., a value close to 0, represents a difficult sample. Note that the heuristic $\mathcal{D}(t)$ necessarily depends on the type of loss function employed: for pointwise loss, it estimates the difficulty for assigning the relevance score s to $\langle \mathbf{q}, \mathbf{d} \rangle$, while, for pairwise loss, it estimates the difficulty of scoring the relevant document pair $\langle \mathbf{q}, \mathbf{d}^+ \rangle$ above the non-relevant pair $\langle \mathbf{q}, \mathbf{d}^- \rangle$.

In our CL framework, during the first learning iteration, training samples are weighted according only to the difficulty function. To ease into the difficult samples, we employ a hyper-parameter m, which represents the training iteration at which to start to give every training sample equal weights.[10] Between the start of training and the mth training iteration, we linearly degrade the importance of the difficulty heuristic. More formally, we define the iteration-informed training sample weight $W_{\mathcal{D}}(t, i)$ given the training iteration i (0-based) as:

$$W_{\mathcal{D}}(t, i) = \begin{cases} \mathcal{D}(t) + \frac{i}{m}\big(1 - \mathcal{D}(t)\big) & i < m \\ 1 & i \geq m \end{cases} \tag{3.9}$$

We then define a new $\mathcal{D}$-informed loss function by including the iteration-informed weight into the standard pointwise or pairwise loss function:

$$\mathcal{L}_{\mathcal{D}}(t, i) = W_{\mathcal{D}}(t, i)\, \mathcal{L}(t) \tag{3.10}$$

Difficulty Heuristics

In a re-ranking setting, a simple source of difficulty information can come from the initial ranking of the documents. Probability ranking models like BM25 rely on term frequency and inverse document frequency to score documents. These characteristics should generally be easy for models to learn because they can learn to identify term

[10] We explore the importance of eventually converging to equal weights in Section 3.5.3.

frequency information (either directly, as is done by models like DRMM and KNRM, or implicitly, as is done by models like BERT through self-attention) and inverse document frequency, e.g., by down-weighting the importance of frequent terms. We postulate that it is inherently more difficult to perform semantic matching needed for identifying documents that have lower initial ranking scores. These scores are also easy to obtain, as they are readily available in a re-ranking setting. Thus, we use unsupervised ranking scores as the basis for our curriculum learning heuristics.

Reciprocal rank heuristic We define $\mathcal{D}_{recip}$ as a measure of difficulty from the reciprocal of the rank at which answers appear in a ranked list. We assume that an answer placed higher compared to the other retrieved answers is "easier" for the ranker to place in that position. A high rank makes relevant documents easier and non-relevant documents harder. In the pointwise setting, relevant documents with a high reciprocal rank are considered "easier" than relevant documents with a low reciprocal rank because the unsupervised ranker assigned a higher score. Conversely, non-relevant documents with a high rank are considered "harder" than samples that are assigned a low rank. Given $\mathbf{d}$ from a set of ranked documents $\mathbf{D}$ for query $\mathbf{q}$ we have:

$$recip_{\mathbf{q},\mathbf{D}}(\mathbf{d}) = \frac{1}{rank_{\mathbf{q},\mathbf{D}}(\mathbf{d})} \tag{3.11}$$

With these conditions in mind, we define $\mathcal{D}_{recip}$ for pointwise loss as:

$$\mathcal{D}^{point}_{recip}(\mathbf{q}, \mathbf{d}, s) = \begin{cases} recip_{\mathbf{q},\mathbf{D}}(\mathbf{d}) & s > 0 \quad \triangleright \textit{relevant} \\ 1 - recip_{\mathbf{q},\mathbf{D}}(\mathbf{d}) & s \leq 0 \quad \triangleright \textit{non-relevant} \end{cases} \tag{3.12}$$

For pairwise loss, we define pairs that have a large difference between the reciprocal ranks to be very difficult (when the non-relevant document is higher) or very easy (when the relevant document is higher). When the reciprocal ranks are similar, we define the difficulty as moderate, with a difficulty close to 0.5. This is accomplished

by taking the difference between the scores and scaling the result within the range $[0, 1]$:

$$\mathcal{D}^{pair}_{recip}(\mathbf{q}, \mathbf{d}^+, \mathbf{d}^-) = \frac{recip_{\mathbf{q},\mathbf{D}}(\mathbf{d}^+) - recip_{\mathbf{q},\mathbf{D}}(\mathbf{d}^-) + 1}{2} \tag{3.13}$$

Normalized score heuristic An alternative to using the ranks of documents by an unsupervised ranker is to use the scores from these rankers. We define $\mathcal{D}_{norm}$ as a measure of difficulty that uses the ranking score information. This allows documents that receive similar (or identical) scores to be considered similarly (or identically) in terms of difficulty. In the case of identical scores, $\mathcal{D}_{norm}$ allows to improve the reproducibility of the CL approach compared to curricula that rely on rank [99]. To account for various ranges in which ranking scores can appear, we apply min-max normalization by query to fit all scores into the $[0, 1]$ interval, eliminating per-query score characteristics. The integration of the normalized score $norm_{\mathbf{q},\mathbf{D}}(\mathbf{d})$ into both pointwise and pairwise rankers are similar to that of the reciprocal rank curriculum:

$$\mathcal{D}^{point}_{norm}(\mathbf{q}, \mathbf{d}, s) = \begin{cases} norm_{\mathbf{q},\mathbf{D}}(\mathbf{d}) & s > 0 \quad \triangleright \textit{ relevant} \\ 1 - norm_{\mathbf{q},\mathbf{D}}(\mathbf{d}) & s \leq 0 \quad \triangleright \textit{ non-relevant} \end{cases} \tag{3.14}$$

$$\mathcal{D}^{pair}_{norm}(\mathbf{q}, \mathbf{d}^+, \mathbf{d}^-) = \frac{norm_{\mathbf{q},\mathbf{D}}(\mathbf{d}^+) - norm_{\mathbf{q},\mathbf{D}}(\mathbf{d}^-) + 1}{2} \tag{3.15}$$

Kernel Density Estimation (KDE) heuristic The normalized score heuristic provides weighting based on ranking score, but it fails to acknowledge an important characteristic of ranking score values: they are often non-linear. For example, it is common for a handful of scores to be comparatively very high, with a long tail of lower scored answers (e.g., with fewer query term matches, see Figure 3.5(a)). We hypothesize that it may be valuable to provide a greater degree of value distinction between scores in areas of high score density (e.g., in the long tail, around a score of 16 and below in

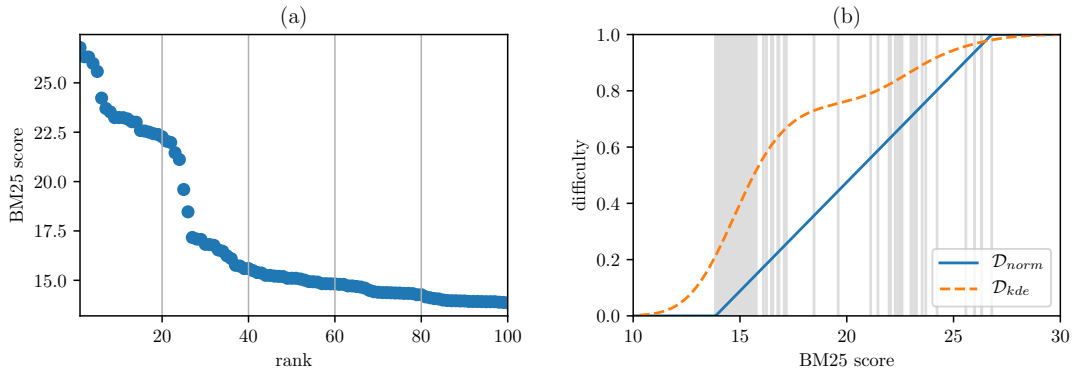

Figure 3.5 Illustration of intuition for using the KDE difficulty heuristic. (a) Example of BM25 scores exhibiting non-linear behavior; there are several answers with a much higher score than others and a long tail of lower-scored answers. (b) Comparison between normalized score (solid blue) and KDE (dashed green) heuristic values by BM25 score. The grey vertical lines indicate the values from the initial ranking (from (a)). Scores are from MS-MARCO query 1000009 retrieved using Anserini's [217] BM25 implementation.

Figure 3.5(a)) and areas with relatively low score density (e.g., around a score of 20). To this end, we construct a Gaussian Kernel Density Estimation (KDE), with the bandwidth selected using Scott's Rule [182]. We then define $\mathcal{D}_{kde}$ by using the CDF of the kernel as the basis of difficulty measure:

$$\mathcal{D}^{point}_{kde}(\mathbf{q}, \mathbf{d}, s) = \begin{cases} KDE_{\mathbf{q},\mathbf{D}}(\mathbf{d}) & s > 0 \quad \triangleright \textit{ relevant} \\ 1 - KDE_{\mathbf{q},\mathbf{D}}(\mathbf{d}) & s \leq 0 \quad \triangleright \textit{ non-relevant} \end{cases} \tag{3.16}$$

$$\mathcal{D}^{pair}_{kde}(\mathbf{q}, \mathbf{d}^+, \mathbf{d}^-) = \frac{KDE_{\mathbf{q},\mathbf{D}}(\mathbf{d}^+) - KDE_{\mathbf{q},\mathbf{D}}(\mathbf{d}^-) + 1}{2} \tag{3.17}$$

where $KDE_{\mathbf{q},\mathbf{D}}(\mathbf{d})$ yields the CDF score of the kernel density estimation for $\mathbf{d}$. An example of the difference between $\mathcal{D}_{norm}$ and $\mathcal{D}_{kde}$ for a particular query is shown in

Table 3.9 Dataset statistics for curriculum learning experiments

Dataset	# Answers	Train Queries (judg. per query)	Validation Queries (judg. per query)	Test Queries (judg. per query)	Test Judgments
TREC DL [132]	8.8M	504k (1.1)	200 (0.7)	43 (215.3)	Human (graded)
TREC CAR [46]	30M	616k (4.8)	2.2k (5.5)	2.4k (6.7)	Inferred (positive only)
ANTIQUE [62]	404k	2.2k (11.3)	200 (11.0)	200 (33.0)	Human (graded)

The values in parentheses indicate the average number of relevance judgments per query.

Figure 3.5(b). This approach has the added benefit of allowing a non-zero difficulty for positive samples that are not retrieved in the ranking list.

In summary, we propose a curriculum learning approach for answer ranking. The approach weights training samples by predicted difficulty. We propose three heuristics for estimating training sample difficulty, based on the rank or score of an unsupervised ranking model.

3.5.3 Experiment

We conduct experiments on three large-scale answer ranking datasets — namely TREC Deep Learning (DL) [34] (Section 3.5.3), TREC Complex Answer Retrieval (CAR) [46] (Section 3.5.3), and ANTIQUE [62] (Section 3.5.3) — and two neural rankers (Vanilla BERT and ConvKNRM) to answer the following research questions:

RQ1 Are the proposed training curricula effective for training neural rankers for answer ranking? (Sections 3.5.3–3.5.3)

RQ2 Under which conditions is each curriculum more effective (e.g., amount and quality of training data, type of neural ranker trained, etc.)? (Sections 3.5.3–3.5.3)

RQ3 Is it important to shift to difficult samples, or can a ranker be successfully trained focusing only on easy samples? (Section 3.5.3)

RQ4 Is focusing on the easy samples first more beneficial to training than focusing on the hardest samples first? (Section 3.5.3)

Each dataset exhibits different characteristics (summarized in Table 3.9). We test using the the Vanilla BERT ranker (from Section 3.2) and ConvKNRM [37]. Based on preliminary experiments that showed that the ConvKNRM model fails to converge when trained using pointwise loss, we only test using pairwise loss. We train both models using the Adam optimizer and a learning rate of 2×10^{-5} for Vanilla BERT and 10^{-3} for ConvKNRM. Furthermore, we use the score additivity technique from [218].

We train each model using training iterations consisting of 32 batches of 16 training samples uniformly selected over the re-ranking pool. We employ gradient accumulation when a training batch is unable to fit on a GPU (e.g., Vanilla BERT models). After each training iteration, the validation performance is assessed. We employ early stopping after 15 consecutive epochs with no improvement to the dataset-dependent validation metric. When training is early stopped, the model is rolled back to the version of that achieved a performance improvement. This yielded up to 130 training iterations. We test our three proposed training curricula ($\mathcal{D}_{recip}$, $\mathcal{D}_{norm}$, and $\mathcal{D}_{kde}$) on each of the datasets and neural rankers. We optimize the parameter m i.e., end of curriculum learning epoch, by fine-tuning on the validation set. For each dataset, ranker, and loss combination, we test $m \in \{1, 5, 10, 20, 50, 100\}$. To put performance of the neural rankers in context, we include the ranking effectiveness of Anserini's [217] implementation of BM25 and SDM [125], both with default parameters, tuned on the validation set ('Tuned'), and tuned on the test set (representing the optimal settings

of parameters for this model, 'Optimized').[11] We also include relevant prior reported results and the optimal re-ranking of the results (i.e., sorting the original ranking list by relevance score, serving as an upper bound to re-ranking performance).

Web Passage Answer Ranking

We first demonstrate the effectiveness of our training curricula on the TREC Deep Learning (DL) 2019 answer passage ranking dataset, which uses the MS-MARCO collection and queries [132]. The training data for this dataset consists of over a million questions collected from the Bing query log. A human annotator was presented a question and a list of 10 candidate answer passages. The annotator was asked to produce a written answer to these questions based on the passages and to indicate the passages that were most valuable in the production of this answer. For the purposes of passage ranking, these passages are considered relevant to the corresponding question. We note that this does not necessarily mean that all correct passages are annotated as relevant, nor it means that the *best* passage is annotated (better answers could exist beyond the 10 shown to the annotator). To overcome this limitation, the TREC DL track manually judged the top retrieved passages for a subset of the test collection. This evaluation setting, which uses manual relevance judgments, is more suitable for evaluation than prior works that relied on incomplete relevance judgments (e.g., [135]). These incomplete training relevance labels also make this dataset suitable for our curriculum learning approach; answers ranked highly by an unsupervised ranker may be relevant, so down-weighting these samples during training may be beneficial.

We train our models using the official MS-MARCO list of training positive and negative relevance judgments. We use a held-out set of 200 queries for validation. We

[11] Models tuned using a grid search: BM25 $k_1 \in [0.1, 4.0]$ by 0.1 and $b \in [0.0, 1.0]$ by 0.05; SDM term, ordered and unordered weights $\in [0, 1]$ by 0.1.

re-rank the official initial test set ranking,[12] and we use the official TREC DL manual relevance judgments for our evaluation and analysis. Statistics about the training, development, and test sets are given in Table 3.9.

Since this work focuses on re-ranking, we evaluate using precision-oriented metrics, and leave recall to future work. We use mean reciprocal rank at 10 (MRR@10) as the validation metric, as it is the official task evaluation metric. Although not included as an official task metric, we also evaluate using P(recision)@1, which indicates the performance of the ranker in a realistic setting in which a single answer is given to a question.

We present the ranking performance for TREC DL in Table 3.10. We observe that under all conditions, our proposed curricula out-perform the ranker when trained without a curriculum for both MRR and P@1 metrics. $\mathcal{D}_{recip}$ outperforms the other curricula for ConvKNRM and pointwise Vanilla BERT, while $\mathcal{D}_{kde}$ outperforms the other curricula for pairwise Vanilla BERT.

When the model significantly under-performs well-tuned BM25 and SDM (ConvKNRM), we observe that the curricula can improve the ranking performance to approximately the level of these baselines. When the model is already doing substantially better (Vanilla BERT), our training curricula also yield a considerable boost to ranking effectiveness. The observation that our approach can improve the ranking effectiveness in both these cases is encouraging, and suggests that the approach is generally beneficial. When compared to the top TREC DL re-ranking results [34], our approach performs favorably. Specifically, the top approach, namely pointwise Vanilla BERT with $\mathcal{D}_{recip}$, ranks second among the submissions. It is only narrowly

[12] Another evaluation setting for TREC DL is "full ranking", in which systems perform initial retrieval in addition to re-ranking. Since this work focuses on improving the effectiveness of re-ranking models rather than initial stage retrieval, we compare with other re-ranking submissions.

exceeded by a much more expensive and complicated approach of pretraining a new

Table 3.10 Ranking performance on the TREC DL 2019 answer passage ranking task.

	Ranker	Training	MRR@10	P@1
	ConvKNRM	Pairwise	0.6159	0.4419
		w/ $\mathcal{D}_{recip}$	**0.6834**	**0.5581**
		w/ $\mathcal{D}_{norm}$	0.6514	0.5116
		w/ $\mathcal{D}_{kde}$	0.6475	0.5116
	Vanilla BERT	Pointwise	0.8740	0.7907
		w/ $\mathcal{D}_{recip}$	**0.8942**	**0.8372**
		w/ $\mathcal{D}_{norm}$	0.8895	0.8140
		w/ $\mathcal{D}_{kde}$	0.8857	0.8140
	Vanilla BERT	Pairwise	0.8477	0.7442
		w/ $\mathcal{D}_{recip}$	0.8624	0.7674
		w/ $\mathcal{D}_{norm}$	0.8581	0.7907
		w/ $\mathcal{D}_{kde}$	**0.8837**	$\uparrow$ **0.8372**
Baselines	BM25	Default	0.7024	0.5814
		Tuned	0.6653	0.5349
		Optimized	0.7555	0.6744
	SDM	Default	0.6276	0.4884
		Tuned	0.6243	0.4884
		Optimized	0.6667	0.5814
	Top TREC Re-Ranking runs [34]	1.	0.907	-
		2.	0.882	-
		3.	0.870	-
	Optimal Re-Ranker		0.9767	0.9767

Significant improvements in performance when using the training curricula (as compared to no curriculum) are indicated with $\uparrow$ (paired t-test $p < 0.05$). There are no statistically-significant differences among the curricula. The top result for each model are listed in bold.

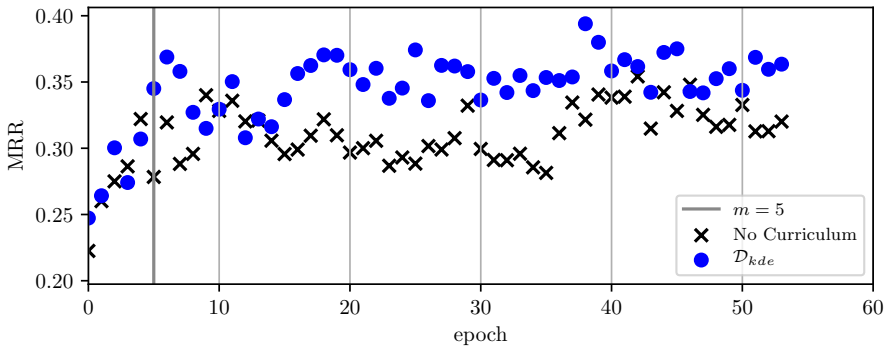

Figure 3.6 Validation performance comparison between Vanilla BERT model trained with and without a curriculum. Training conducted using pointwise loss without a curriculum (black x) and with the $\mathcal{D}_{kde}$ curriculum (blue circle) for TREC DL. The tuned m parameter for the $\mathcal{D}_{kde}$ curriculum used here is marked with a vertical line. While the variant without a curriculum quickly reaches optimal performance, the curriculum approach reaches a higher performance faster and offers a stronger foundation on which to continue training after the curriculum terminates.

BERT model from scratch using a different training objective. Our results indicate that this can be avoided by simply doing a better job weighting the training samples.

To gain a better understanding of how the curriculum benefits the training process, we compare the validation performance of the pointwise Vanilla BERT model with the $\mathcal{D}_{kde}$ training curriculum to the same model when trained without a curriculum (Figure 3.6). This reveals that when not using a curriculum, the validation performance peaks early, suggesting that it is overfitting to difficult examples. The curriculum, however, has even stronger early performance and is in a better position to incorporate difficult samples as training continues. Note that the tuned end of curriculum epoch is $m = 5$ for this example, showing that the curriculum does not need to be in place for long to get these benefits. Also note that the training data were

presented in the exact same order in both cases, showing the importance of weighting the loss effectively.

Complex Answer Passage Ranking

We also evaluate our curriculum learning framework on the TREC Complex Answer Retrieval (CAR) dataset [46]. To compare with prior work, we use version 1.0 of the dataset. This dataset consists of topics in the form of a hierarchy of article headings (e.g., *Green Sea Turtle » Ecology and behavior » Diet*). A standard set of automatically-generated relevance judgments are provided by assuming paragraphs (passages) under a heading are relevant to the query corresponding to the heading. The automatic relevance assessments provide a large amount of training data, but can suffer from variable quality (e.g., some paragraphs are very difficult to match as they provide little context). This makes TREC CAR a good application of training curricula; it contains many positive relevance samples that are difficult to match. A set of manually-graded relevance assessments are also provided by TREC assessors. However, due to the shallow assessment pool used (due to the large number of topics), we opt to only evaluate our approach using the automatic judgments.[13] We use TREC 2017 (`Y1`) training data with `hierarchical` relevance judgments. We also compare our results to the performance reported by [135] and [111], which use BERT and the PACRR neural ranking architecture augmented with entity embeddings for classification, respectively.

Following previous work [135], we train and validate our models using the top 10 results retrieved by BM25 and test on the top 1000 results. We use the official task metric of R-Prec(ision) to validate our model. We also report MAP, another official

[13]The track report suggests that the automatic judgments are a reasonable proxy for manual judgments as there is a strong correlation between the automatic and manual performance among the TREC submissions [46].

Table 3.11 Ranking performance on the TREC CAR complex answer passage ranking task.

	Ranker	Training	R-Prec	MAP
	ConvKNRM	Pairwise	0.1081	0.1412
		w/ $\mathcal{D}_{recip}$	↑ 0.1174	↑ 0.1493
		w/ $\mathcal{D}_{norm}$	↑ **0.1258**	↑ **0.1572**
		w/ $\mathcal{D}_{kde}$	↑ 0.1227	↑ 0.1553
	Vanilla BERT	Pointwise	0.2026	0.2490
		w/ $\mathcal{D}_{recip}$	↑ **0.2446**	↑ **0.2864**
		w/ $\mathcal{D}_{norm}$	↑ 0.2370	↑ 0.2764
		w/ $\mathcal{D}_{kde}$	↑ 0.2370	↑ 0.2795
	Vanilla BERT	Pairwise	0.2731	0.3207
		w/ $\mathcal{D}_{recip}$	↑ 0.2914	↑ 0.3298
		w/ $\mathcal{D}_{norm}$	↑ **0.2921**	↑ **0.3307**
		w/ $\mathcal{D}_{kde}$	↑ 0.2844	0.3254
Baselines	BM25	Default Settings	0.1201	0.1563
		Tuned	0.1223	0.1583
		Optimized	0.1231	0.1588
	SDM	Default Settings	0.1154	0.1463
		Tuned	0.1099	0.1420
		Optimized	0.1155	0.1459
	BERT Large [135]		-	0.335
	BERT Base [135]		-	0.310
	PACRR [111]		0.146	0.176
	Optimal Re-Ranker		0.6694	0.6694

Significant improvements in performance when using the training curricula (as compared to no curriculum) for each model are indicated with ↑ (paired t-test $p < 0.05$, no significant reductions observed). For Pointwise loss, $\mathcal{D}_{recip}$ significantly outperforms $\mathcal{D}_{norm}$ in terms of MAP. There are no other significant differences among the training curricula. The top results in each section are indicated in bold.

metric for the task. We use these metrics rather than MRR and P@1 because CAR queries often need many relevant passages to answer the question, not just one.

We present the performance of our training curricula on TREC CAR in Table 3.11. We observe that in all cases, the training curricula significantly improve the ranking effectiveness. When training rankers using pairwise loss, the $\mathcal{D}_{norm}$ curriculum is most effective, and when training with pointwise loss, the $\mathcal{D}_{recip}$ curriculum is most effective. In the case of ConvKNRM, without the curriculum, the ranker under-performs the unsupervised BM25 and SDM baselines; with the curricula, it performs on-par with them. For Vanilla BERT, both when trained with pairwise and pointwise losses, the ranker outperforms the unsupervised baselines without the curricula, and improves significantly when using the curricula.

When compared with the supervised baselines, i.e., BERT and PACRR, the Vanilla BERT model trained with pairwise loss and $\mathcal{D}_{norm}$ curriculum ends up performing about as well as the large BERT baseline reported by [135] (0.3307 versus 0.335 in terms of MAP, no statistically significant difference). This is a considerable achievement because the Vanilla BERT model is half the size and about twice as fast to execute. This observation strengthens the case for using curricula when training because it can allow for similar gains as using a much larger model.

The remaining gap between our trained models and the optimal re-ranker on the CAR dataset, however, indicates that there is still room for improvement in this task. In particular, a considerable challenge is ranking passages without much context highly without adding too much noise to the model.

Non-factoid Question Answering

We also test our approach on the ANTIQUE non-factoid question answering dataset [62]. Unlike TREC DL and CAR, ANTIQUE has more thoroughly anno-

Table 3.12 Ranking performance on the ANTIQUE non-factoid question answering task.

	Ranker	Training	MRR	P@1
	ConvKNRM	Pairwise	0.4920	0.3650
		w/ $\mathcal{D}_{recip}$	↑ **0.5617**	↑ **0.4550**
		w/ $\mathcal{D}_{norm}$	↑ 0.5523	↑ 0.4450
		w/ $\mathcal{D}_{kde}$	↑ 0.5563	↑ 0.4500
	Vanilla BERT	Pointwise	0.6694	0.5550
		w/ $\mathcal{D}_{recip}$	0.6858	0.5850
		w/ $\mathcal{D}_{norm}$	0.6888	0.5800
		w/ $\mathcal{D}_{kde}$	**0.6953**	**0.6000**
	Vanilla BERT	Pairwise	0.6999	0.5850
		w/ $\mathcal{D}_{recip}$	↑ **0.7335**	↑ **0.6450**
		w/ $\mathcal{D}_{norm}$	0.7237	0.6250
		w/ $\mathcal{D}_{kde}$	0.7244	0.6250
Baselines	BM25	Default Settings	0.5464	0.4450
		Tuned	0.5802	0.4550
		Optimized	0.6035	0.4950
	SDM	Default Settings	0.5229	0.4050
		Tuned	0.5377	0.4400
		Optimized	0.5491	0.4700
	Best prior published (BERT) [62]		0.7968	0.7092
	Optimal Re-Ranker		0.9400	0.9400

Significant improvements in performance when using the training curricula (as compared to no curriculum) are indicated with ↑ (paired t-test $p < 0.05$). There are no statistically-significant differences among the curricula. The top results in each section are indicated in bold.

tated training queries, with an around 11 graded relevance judgments per query in the training and validation collections (crowdsourced) (see Table 3.9). Furthermore, these include explicit labels for non-relevant answers, which are not present in the other two datasets. This more extensive annotation comes at the expense of scale, however, with far fewer queries to train upon. Nevertheless, ANTIQUE represents another valuable set of conditions under which to evaluate our curricula. We randomly sample from the top 100 BM25 results for additional negative samples during training. We validate and test by re-ranking the top 100 BM25 results, and MRR as the validation metric and P@1 as a secondary metric. We use these two official task metrics (at relevance level of 3 or higher, as specified in [62]) because the answers in ANTIQUE are self-contained, and these metrics emphasize correct answers that are ranked highly and first, respectively.

We report the curricula performance on ANTIQUE in Table 3.12. Similar to TREC DL, we observe that the $\mathcal{D}_{recip}$ and $\mathcal{D}_{kde}$ curricula are the most effective. For ConvKNRM, the curricula were able to overcome what would otherwise be a model that under-performs w.r.t. the BM25 and SDM unsupervised baselines. For the pointwise and pairwise Vanilla BERT models (which are already very effective), we observe gains beyond. In the case of pairwise-trained Vanilla BERT, the $\mathcal{D}_{recip}$ curriculum significantly boosted ranking performance. Despite our efforts to reproduce the effectiveness of BERT reported in [62], we were unable to do so using the experimental settings described in that work. These results are still below that of an optional re-ranking, suggesting that there is still considerable room for improvement when ranking these non-factoid answers.

To answer RQ1 (whether the training curricula are effective), we observed that for three answer ranking datasets (TREC DL, TREC CAR, and ANTIQUE) these curricula can improve the ranking effectiveness across multiple neural rankers and

loss functions. We observe that when a ranker initially underperforms standard baselines (e.g., ConvKNRM), the performance is effectively boosted to the level of those baselines. When the ranker already exceeds these baselines (e.g., Vanilla BERT), we also observe a boost to ranking effectiveness, often comparable to or approaching the state-of-the-art while being considerably faster (e.g., using BERT Base instead of BERT Large) or less complicated (e.g., not requiring an expensive pre-training step). The observation that the curricula are effective in these various conditions suggests that these curricula are generally effective. To answer RQ2 (under what conditions each curriculum is effective), we observe that $\mathcal{D}_{recip}$ and $\mathcal{D}_{kde}$ are generally more effective for natural-language questions (TREC DL and ANTIQUE), while $\mathcal{D}_{norm}$ is more effective for keyword/structured questions (TREC CAR). One possible alternative explanation may be that the latter is better with weak relevance labels, as TREC CAR's relevance labels are obtained through a heuristic, rather than human annotators. It does not appear as if the amount of training data has an effect, as TREC DL and ANTIQUE exhibit similar characteristics, while having drastically different amounts of training data.

End of Curriculum Evaluation

We already observed in Figure 3.6 that when using a training curriculum, ranking performance not only peaks higher sooner, but also leaves the model in a better starting point for when all samples are weighted equally. However, an important question remains: Is it important to train with equal weight for all samples or can the difficulty weights be used exclusively? To this end, we perform a test that forgoes the curriculum convergence parameter m, directly using $\mathcal{D}(\cdot)$ as the training sample weight, regardless of training iteration (i.e., $m = \infty$, or equivalently $W = \mathcal{D}$ instead of Eq. 3.9).

Table 3.13 Ranker performance when the curriculum always uses difficulty scores, and when employing the anti-curriculum.

TREC DL

Ranker	Curriculum	m	MRR@10	P@1
ConvKNRM	$\mathcal{D}^{pair}_{recip}$	20	**0.6834**	**0.5581**
	$\mathcal{D}^{pair}_{recip}$	∞	0.6744	0.5581
	$\widehat{\mathcal{D}}^{pair}_{recip}$	20	$\downarrow$ 0.5414	$\downarrow$ 0.3721
Vanilla BERT	$\mathcal{D}^{point}_{recip}$	10	**0.8942**	**0.8372**
	$\mathcal{D}^{point}_{recip}$	∞	0.8205	0.7209
	$\widehat{\mathcal{D}}^{point}_{recip}$	10	0.8527	0.7442
Vanilla BERT	$\mathcal{D}^{pair}_{kde}$	20	**0.8837**	**0.8372**
	$\mathcal{D}^{pair}_{kde}$	∞	$\downarrow$ 0.7752	$\downarrow$ 0.6279
	$\widehat{\mathcal{D}}^{pair}_{kde}$	20	0.8314	0.7209

TREC CAR

Ranker	Curriculum	m	R-Prec	MAP
ConvKNRM	$\mathcal{D}^{pair}_{norm}$	50	**0.1258**	0.1572
	$\mathcal{D}^{pair}_{norm}$	∞	0.1250	**0.1579**
	$\widehat{\mathcal{D}}^{pair}_{norm}$	50	$\downarrow$ 0.1030	$\downarrow$ 0.1324
Vanilla BERT	$\mathcal{D}^{point}_{recip}$	20	0.2446	0.2864
	$\mathcal{D}^{point}_{recip}$	∞	**0.2475**	**0.2894**
	$\widehat{\mathcal{D}}^{point}_{recip}$	20	$\downarrow$ 0.2258	$\downarrow$ 0.2709
Vanilla BERT	$\mathcal{D}^{pair}_{norm}$	10	**0.2921**	**0.3307**
	$\mathcal{D}^{pair}_{norm}$	∞	$\downarrow$ 0.2669	$\downarrow$ 0.3103
	$\widehat{\mathcal{D}}^{pair}_{norm}$	10	0.2837	0.3276

ANTIQUE

Ranker	Curriculum	m	MRR	P@1
ConvKNRM	$\mathcal{D}^{pair}_{recip}$	100	**0.5617**	**0.4550**
	$\mathcal{D}^{pair}_{recip}$	∞	0.5368	0.4100
	$\widehat{\mathcal{D}}^{pair}_{recip}$	100	0.5366	0.4200
Vanilla BERT	$\mathcal{D}^{point}_{kde}$	10	**0.6953**	**0.6000**
	$\mathcal{D}^{point}_{kde}$	∞	$\downarrow$ 0.6139	$\downarrow$ 0.4750
	$\widehat{\mathcal{D}}^{point}_{kde}$	10	0.6677	0.5500
Vanilla BERT	$\mathcal{D}^{pair}_{recip}$	5	**0.7335**	**0.6450**
	$\mathcal{D}^{pair}_{recip}$	∞	0.7158	0.6150
	$\widehat{\mathcal{D}}^{pair}_{recip}$	5	0.7193	0.6200

Significant reductions in performance are indicated with $\downarrow$ (paired t-test, $p < 0.05$).

We report the performance for this experiment on each dataset for each top-performing curriculum in Table 3.13 ($m = \infty$ setting). We observe that for all models on the TREC DL and ANTIQUE datasets, this approach leads to a drop in ranking effectiveness, suggesting that it is important to eventually perform equal sample weighting. Intuitively, this is important because if easy samples are always weighted higher than difficult samples, the model will be hindered in learning the more complicated function to rank difficult samples. Curiously, for TREC CAR, this setting sometimes leads to improved ranking effectiveness (though not a statistically significant improvement). One possible explanation is that in situations where weak labels are used (rather than human-judged labels from top retrieved results), it may be better to always apply the weighting, as some inferred positive labels may be too distant from what the model will typically encounter at inference time.

To answer RQ3 (whether shifting to difficult samples is important), we find that it is indeed beneficial to use our proposed weighting technique given in Eq. 3.9, rather than always applying the difficulty weighting when using manually-assessed relevance labels.

Anti-curriculum: Hardest Samples First

To test whether our intuitions that "difficult" samples are harmful during early phases of training, we conduct a study using an anti-curriculum, i.e., we train our models by weighting the more difficult samples higher than the easier samples. This was applied by swapping out the difficulty function $\mathcal{D}$ with $\widehat{\mathcal{D}}(\cdot) = 1 - \mathcal{D}(\cdot)$. This has the effect of assigning high weights to samples that previously had low weights and vice versa. All usage of the difficulty function remains unchanged (e.g., the integration of the difficulty function into the weight function).

Table 3.13 ($\widehat{\mathcal{D}}$ setting)presents a ranking performance comparison when using the anti-curriculum. We observe that the anti-curriculum always reduces ranking effectiveness, sometimes significantly. In some cases, this can be rather severe; on TREC DL for Vanilla BERT (pairwise), the MRR is reduced by 0.0523 and P@1 is reduced by 0.1163, resulting in a model that underperforms one without any weighting at all. To answer RQ4, these results suggest that there is benefit to weighting the easiest samples higher first, rather than the more difficult samples.

3.5.4 Summary

In this section, we demonstrated that dataset statistics can be incorporated into the ranking model training process. Specifically, we use statistics from unsupervised lexical models to estimate difficulty, and use this estimated difficulty to weight training samples. This approach is in contrast with using dataset statistics as model features (as was shown in Section 2.3), and provides further evidence in support of Hypothesis 1.2. There are practical advantages to using dataset statistics in the training process rather than as model features. As we discussed in Section 2.2 and Section 3.4, it can be beneficial to transfer relevance signals across datasets and tasks. Models that do not rely on dataset-specific features can more easily be used in this setting. We further explore the task of relevance transfer in the following section.

3.6 Searching COVID-19 Literature

The emergence of the Severe Acute Respiratory Syndrome Coronavirus 2 (SARS-CoV-2) prompted a worldwide research response. In the first 120 days of 2020, researchers published over 10,000 articles related to SARS-CoV-2 or COVID-19. Together with articles about similar viruses researched before 2020, the body of research approaches

60,000 articles. Such a large body of research results in a considerable burden for those seeking information about various facets of the virus, including researchers, clinicians, and policy-makers.

In this section, we explore techniques for transferring relevance signals across domains to improve the search over COVID-19 literature. These experiments differ from those explored in Section 2.2 in that we perform tests using rankers that use contextualized language models. Furthermore, we use the MS-MARCO dataset for training, rather than the New York Times Corpus or Wikipedia, as this resource is widely available and is more similar to the target task. Similar to Section 2.2, we find that techniques for filtering the training data are effective. Finally, this differs from the cross-lingual work explored in Section 3.4 because the domains are different and the language is the same; Section 3.4 explored transferring between different languages (English to Chinese, Arabic, and Spanish) on the same domain (news content), whereas this section explores transferring between different domains (web passages and scientific literature).

In this section, we introduce an approach called SLEDGE: a simple yet effective zero-shot baseline for coronavirus Scientific knowLEDGE search. SLEDGE builds upon the Vanilla BERT model introduced in Section 3.2 for COVID-19 search with three simple techniques. First, we propose a training data filtering technique to help the ranking model learn relevance signals typical in medical text. The training data we use comes entirely from another dataset (MS-MARCO, Campos et al. [19]), resulting in our model being zero-shot. Since MS-MARCO is a large collection of real user queries (over 800,000), it allows us to filter aggressively and still have adequate training data. Second, we replace the general contextualized language model BERT with one pre-trained on scientific literature (SciBERT, Beltagy et al. [10]). This pre-training prepares the model for the type of language typically seen in scientific articles.

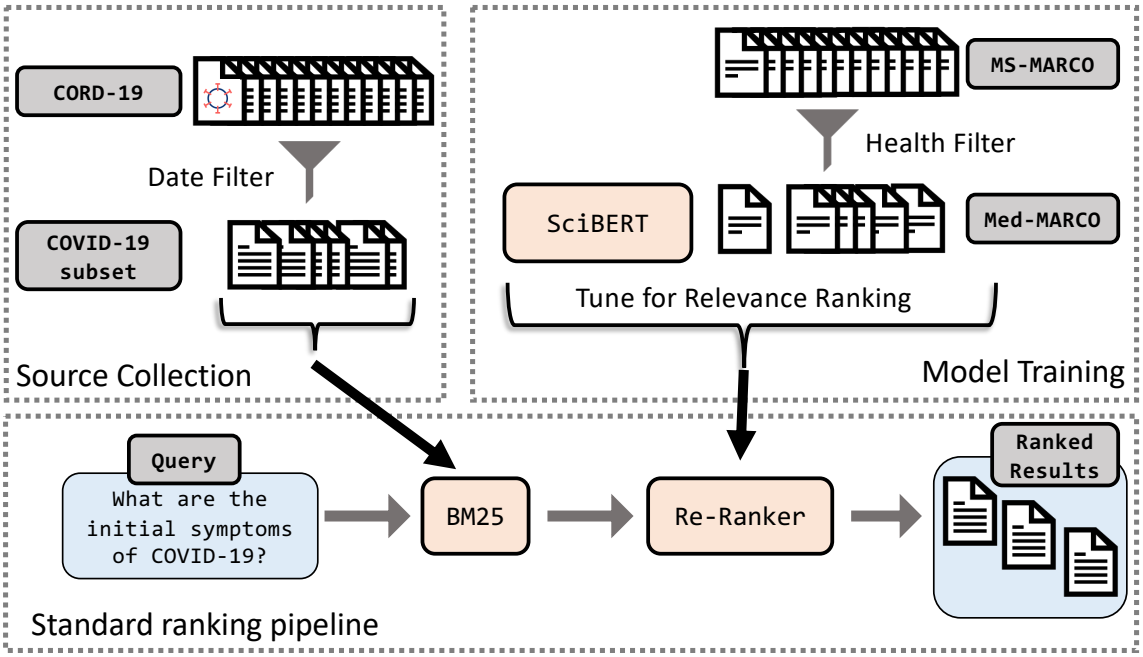

Figure 3.7 Overview of SLEDGE.

Since the document collection (CORD-19, Wang et al. [208]) contains articles about prior viruses, we filter out articles published before 2020 to eliminate less pertinent articles. An overview of this process is shown in Figure 3.7.

We show that each of the techniques mentioned above positively impacts the ranking effectiveness of SLEDGE through an ablation analysis. Our zero-shot approach performs comparably to (or outperforms) top-scoring submissions to the TREC-COVID document ranking shared task [166], a new testbed for evaluating of search methods for COVID-19.

3.6.1 Methodology

To build a ranking model for COVID search, we modify the standard zero-shot Vanilla BERT document re-ranking pipeline [3, 110]. We find that while these modifications are simple, they are effective for maximizing ranking performance. We note that this process neither requires COVID relevance training data nor involves a priori inspection of the queries and their characteristics. Thus, we consider our method zero-shot.

To train in a zero-shot setting, we employ a large dataset of general-domain natural language question and answer paragraphs: MS-MARCO [19]. However, naïve domain transfer is not optimal since most questions in the dataset are not medical-related, causing a domain mismatch between the training and evaluation data. To overcome this challenge, we apply a heuristic to filter the collection to only medical-related questions. The filter removes questions that do not contain terms appearing in the MedSyn [223], a lexicon of layperson and expert terminology for various medical conditions. We manually remove several common terms from the lexicon that commonly introduce queries that are not medical-related. For example, MedSyn includes the term *gas* (referring to the medical concept of flatulence in North American English), commonly also refers to gasoline or natural gas. Note that we made these decisions without considering COVID-19 specifically—only a broad relation to the medical domain. MS-MARCO originally consists of 809K questions. After filtering, 79K of the original questions remain (9.7%). We refer to this subset of MS-MARCO as Med-MARCO. From a random sample of 100 queries from Med-MARCO, 78 were judged by the authors as medical-related, suggesting the filter has reasonable precision. Examples questions from this process include *causes of peritoneal cancer prognosis* and

what is squalene anthrax sleep apnea. We make a list of the query IDs corresponding to Med-MARCO available,[14].

Second, we replace the general-language BERT model with a variant tuned on scientific literature (including medical literature). Specifically, we use SciBERT [10], which has an identical structure as BERT, but was trained on a multi-domain corpus of scientific publications. It also uses a WordPiece lexicon based on the training data, allowing the model to better account for subwords commonly found in scientific text. During model training, we employ the pairwise cross-entropy loss function from Nogueira and Cho [135]. Relevant and non-relevant documents are sampled in sequence from the official MS-MARCO training pair list (filtered down to Med-MARCO queries).

Third, we apply a filter to the document collection that removes any articles published before January 1, 2020. This filter aims to improve the retrieval system's precision by eliminating articles that may discuss other topics. The date was chosen because little was known about COVID-19 prior to 2020, and some documents do not include a full publication date (only a year), making this filter simple to apply. In real-life search engines, date filtering can often be applied at the discretion of the user.

3.6.2 Experiment

We now explore the ranking effectiveness of our approach. We evaluate the performance of SLEDGE using Round 1 and 2. At the time of writing, the only training

[14] `https://github.com/Georgetown-IR-Lab/covid-neural-ir/blob/master/med-msmarco-train.txt`

data available for the task was the Round 1 data. of the TREC-COVID Information Retrieval Benchmark [166].[15] TREC-COVID uses the CORD-19 document collection [208] (2020-05-01 version, 59,943 articles), with a set of 35 topics related to COVID-19. These topics include natural questions such as: *what is the origin of COVID-19* and *how does the coronavirus respond to changes in the weather*. The top articles of participating systems in each round were judged by expert assessors, who rated each article as non-relevant (0), partially-relevant (1), or fully-relevant (2) to the topic. In total, 20,728 relevance judgments were collected (avg. 592 per topic), with 74% non-relevant, 12% partially relevant, and 14% fully-relevant. These rates remained nearly constant between rounds 1 and 2.

We use normalized Discounted Cumulative Gain with a cutoff of 10 (nDCG@10), Precision at 5 of partially and fully-relevant documents (P@5), and Precision at 5 of only fully relevant documents (P@5 (F)). Both nDCG@10 and P@5 are official task metrics; we include the P@5 filtered to only fully-relevance documents because it exposed some interesting trends in our analysis. We also report the percentage of the top 10 documents for each query that have relevance judgments (J@10). In an additional evaluation, we measure the performance using only judged documents to ensure that unjudged documents do not impact our findings. We used `trec_eval`[16] for all metrics. These measures represent a precision-focused evaluation; since re-ranking methods like ours focus on improving precision, we leave recall-oriented evaluations to future work.

[15]Round 2 uses *residual collection* evaluation, meaning that all documents judged in Round 1 are disregarded. Although this is an important setting for building up a dataset and allows for approaches like manual relevance feedback, we feel that this setting does not mimic an actual search engine, especially in the zero-shot setting. Thus, we evaluate on the concatenation of Round 1 and 2 settings and mark the systems that use Round 1 judgments for training or tuning of their system.

[16]`https://github.com/usnistgov/trec_eval`

Our initial ranking is conducted using BM25 with default settings over the full document text to adhere to the zero-shot setting. Re-ranking is conducted over the abstracts only, avoiding the need to perform score aggregation (since BERT models are limited in the document length). We utilize only the natural-language question (ignoring the keyword query and extended narrative). We conduct an ablation that compares SLEDGE to versions using BERT (instead of SciBERT), and the full MS-MARCO dataset (MSM) (rather than the Med-MARCO subset (MedM)). We compare with several baselines under the same evaluation settings.

- **BM25**: the initial BM25 ranking.
- **ConvKNRM**: The convolutional KNRM model [37], trained on MS-MARCO data.
- **CEDR KNRM**: The KNRM model, augmented with contextualized embeddings, as introduced in Section 3.2. The model is trained on MS-MARCO data. We use the `bert-base-uncased` model for the contextualized embeddings.
- **Seq2seq T5**: The text-to-text-transformer (T5) model [159], tuned for ranking by predicting *true* or *false* as the next term in a sequence consisting of the query and document [139].
- **Fusion**: a reciprocal rank fusion method [30] of BM25 over the abstract, full text, and individual paragraphs. Fusion1 uses a concatenation of the keywords and question, and Fusion2 uses the entity extraction technique from the Round 1 `udel` submission.[17]

Our work utilizes a variety of existing open-source tools: OpenNIR [107], Anserini [217], and the HuggingFace Transformers library [211]. We utilize a held-out subset of 200 queries from the MS-MARCO training set as a validation set for the sole purpose of picking the optimal training epoch. The Vanilla BERT and

[17] `https://github.com/castorini/anserini/blob/master/docs/experiments-covid.md`

Table 3.14 Ablation results and comparison of SLEDGE and other zero-shot baselines on TREC-COVID Rounds 1 and 2.

		Including Unjudged				Judged Only		
Model	Training	nDCG@10	P@5	P@5 (F)	J@10	nDCG@10	P@5	P@5 (F)
BM25	-	* 0.368	* 0.469	* 0.331	75%	* 0.436	* 0.520	* 0.383
+ BERT	MSM	* 0.547	* 0.617	* 0.480	83%	* 0.617	* 0.703	* 0.549
+ BERT	MedM	0.625	* 0.697	* 0.571	92%	0.657	* 0.737	* 0.606
+ SciBERT	MSM	0.667	0.754	0.611	88%	**0.724**	* 0.789	0.646
+ SciBERT (SLEDGE)	MedM	**0.681**	**0.800**	**0.663**	90%	0.719	**0.846**	**0.697**
+ ConvKNRM	MSM	0.536	0.617	0.491	86%	0.580	0.645	0.508
+ ConvKNRM	MedM	0.565	0.668	0.525	86%	0.621	0.714	0.565
+ CEDR-KNRM	MSM	0.514	0.617	0.468	86%	0.524	0.628	0.474
+ CEDR-KNRM	MedM	0.619	0.714	0.560	89%	0.649	0.742	0.582
+ Seq2seq T5	MSM	0.656	0.737	0.634	90%	0.685	0.765	0.651
+ Seq2seq T5	MedM	0.626	0.714	0.594	86%	0.678	0.754	0.628
Fusion1	-	0.519	0.640	0.457	94%	0.534	0.640	0.457
Fusion2	-	0.601	0.737	0.565	96%	0.605	0.737	0.565

The top results are shown in bold. SciBERT with MedM (SLEDGE) significantly outperforms values in the top (ablation) section marked with * ($p < 0.05$, paired t-test, Bonferroni correction).

SciBERT models take approximately 3 hours to train/validate, and inference on TREC-COVID takes approximately 15 minutes on modern GPUs. The BERT model has 157M parameters, and the SciBERT model has 158M parameters.

Results

Ranking effectiveness is presented in Table 3.14. We first compare the ablations of our approach (top section). We note that SciBERT significantly ($p < 0.05$, paired t-test, Boneferroni correction) outperforms BM25 and BERT trained on MSM across all metrics. There is a less dramatic jump between BERT MSM and BERT MedM, demonstrating the importance of filtering the training data properly. This is echoed

between SciBERT MSM and SciBERT MedM, though the difference is only significant for P@5 when only considering the judged documents. These results demonstrate the importance of both pre-training on appropriate data and fine-tuning using a proper subset of the larger data. While both yield improvements (that can be additive), the pre-training objective appears to be more impactful, based on the overall better scores of SciBERT.

Compared to baseline systems (bottom section), we observe that SLEDGE offers superior effectiveness. Specifically, we see that ConvKNRM, CEDR-KNRM, and Seq2seq T5 all improve upon the initial BM25 ranking. Training on Med-MARCO (rather than the full MS-MARCO) also improves each of the baselines, except, curiously, Seq2seq T5. This model may benefit from the larger amount of training data the full MS-MARCO dataset offers. Finally, both fusion methods outperform the base BM25 model. However, we note that these models utilize two fields available for each query: the keyword-based query and the full natural-language question text—a luxury not available in practical search environments. (Recall that SLEDGE and the other baselines in Table 3.14 only use the natural-language query.)

We now compare our approach with the top-performing submissions to the TREC COVID shared task (many of which are not zero-shot methods) in Table 3.15. We note that these experimental settings for these runs differ from our main experiments. For instance, `mpiid5_run3` [95] and `SparseDenseSciBERT` use relevant information from Round 1 as training data, and `covidex.t5` uses combined keyword query and natural-language questions. Therefore, these performance metrics are not directly comparable to our zero-shot runs. Despite this, SLEDGE still achieves competitive performance compared to these models. For instance, it consistently scores comparably or higher than `covidex.t5` (includes a more powerful language model, a more effective initial ranking model, and multiple topic fields) and `SparseDenseSciBert` (which uses

Table 3.15 TREC COVID Round 1 and 2 comparison between SLEDGE and other top official Round 2 submissions.

		Including Unjudged				Judged Only		
Model	Training	nDCG@10	P@5	P@5 (F)	J@10	nDCG@10	P@5	P@5 (F)
SLEDGE (ours)	MedM	0.681	0.800	0.663	90%	0.719	0.846	**0.697**
`covidex.t5`†	MSM, MedM	0.618	0.731	0.560	94%	0.643	0.731	0.560
with date filter		0.652	0.760	0.600	92%	0.680	0.777	0.611
`SparseDenseSciBert`†	MedM	0.672	0.760	0.646	96%	0.692	0.760	0.646
with date filter		**0.699**	0.805	**0.691**	94%	**0.724**	0.811	0.691
`mpiid5_run3`†	MSM, Rnd1	0.684	**0.851**	0.640	93%	0.719	**0.851**	0.640
with date filter		0.679	0.834	0.657	90%	0.722	0.834	0.657

We apply the date filter for a more complete comparison. Note that experimental differences exist between our system and these submissions, including the use of multiple topic fields and the utilization of Round 1 training data for training or tuning. The top result is marked in bold.

neural approaches for the initial ranking stage). Our method even performs comparably to the `mpiid5.run3` model, which was trained directly on Round 1 judgments. Interestingly, we observe that our simple baseline approach of re-ranking using T5 strictly with the natural-language question against the paper title and abstract (Seq2seq T5 in Table 3.14) is more effective than the more involved approach employed by `covidex.t5`. When we apply the same date filtering to the official runs, we observe that the differences narrow. We also present SLEDGE topping the Round 1 leaderboard in Table 3.16. We observe again that our model excels at finding highly-relevant documents.

To gain a better understanding of the impact of filtering the document collection to only articles published on or after January 1, 2020, we first compare the performance of SLEDGE with and without the filter. Disregarding unjudged documents, it has an nDCG@10 of 0.668 (−0.051), P@5 of 0.777 (−0.069) and P@5 (F) of 0.589 (−0.108).

Table 3.16 TREC-COVID Round 1 leaderboard (automatic systems).

System	nDCG@10	P@5	P@5 (F)
SLEDGE (ours)	**0.641**	0.747	**0.633**
`sab20.1.meta.docs`	0.608	**0.780**	0.487
`IRIT_marked_base`	0.588	0.720	0.540
`CSIROmedNIR`	0.588	0.660	0.587

SLEDGE outperforms the highest-scoring run in terms of nDCG@10 and P@5 (F).

All these differences are statistically significant. By far the largest reduction is on fully-relevant P@5, meaning that it can be more difficult to find highly relevant documents when considering the full document collection. We observed similar trends for BM25, with and without the 2020 filter. These trends also align with observations we made from the judgments themselves; we find that only 16% of judged documents from prior to 2020 were considered relevant (with only 5% fully relevant). Meanwhile, 32% of judged documents after 2020 were considered relevant (19% fully relevant).

3.6.3 Summary

We demonstrated that relevance signals can be transferred across domains (from web passages to scientific literature), providing further support for Hypothesis 1.1. We showed that although naïve transfer approaches are reasonably effective, selecting more proper training data and a better base model can significantly improve effectiveness. This work also highlights the importance of this line of work, by showing how this line of work can improve information access during a global crisis.

3.7 Discussion and Conclusions

In this chapter, I demonstrated the effectiveness of contextualized language models for neural ranking. I first showed that utilizing the model's classification mechanism alone can produce effective neural ad-hoc ranking models. I then showed that token-level signals from these models can be incorporated into existing neural ranking architectures (such as PACRR, KNRM, and DRMM) effectively. A combination model of both of these techniques also produces an effective ranking model, suggesting that each captures different relevance notions. I also demonstrated how contextualized language models can be adapted to another ranking task, namely the ranking of clinical report versions by level of discrepancy. Together, these validate Hypothesis 1.3. I also demonstrated that one can utilize multi-lingual contextualized language models to overcome limited training data in languages other than English. This validates Hypothesis 1.4. I then re-investigated Hypotheses 1.1 and 1.2 from Chapter 2 when contextualized models are considered. I showed how ranking scores can be used to augment the training process of these models, providing further evidence of Hypothesis 1.2. And finally, I showed how relevance signals can be effectively transferred across domains, providing further evidence of Hypothesis 1.1.

Chapter 4

Computational Efficiency of Contextualized Neural Ranking

Although neural ranking approaches that use contextualized language models can be effective (as demonstrated in Chapter 3), the added computation has a detrimental effect at query-time. Specifically, the need to compute the document scores for a given query can be costly and add to the query-time latency. In this section, I first cover prior work in efficiency of neural ranking (Section 4.1). I then show how pre-computing partial contextualized representations at index-time can reduce the computational cost at query-time with minimal effect on ranking effectiveness (Section 4.2). The pre-computation approach is general and can be applied to a variety of contextualized language models. Then, in Chapter 4.3, I show how a ranking architecture can be designed specifically to take advantage of this quality while reducing storage costs and query-time latency and improving model interpretability.

4.1 Background and Preliminaries

Scalability and computational efficiency are central challenges in information retrieval. While the efficiency of learning to rank solutions for document re-ranking have been extensively studied [41, 91, 197], computational efficiency concerns have largely be ignored by prior work in neural ranking, prompting some to call for more attention to this matter [68, 98]. That being said, some efforts do exist. For instance, Ji et al. [78] demonstrate that Locality-Sensitive Hashing (LSH) and other tricks

can be employed to improve the performance of interaction-focused methods such as DRMM [56], KNRM [214], and ConvKNRM [37]. This approach does not work for transformer models, however, because further processing of the term embeddings is required (rather than only computing similarity scores between the query and document).

As shown in Chapter 3, pretrained transformer networks, such as BERT, can be very beneficial for ranking. However, they are usually characterized by a very large numbers of parameters and very long inference times, making them unusable in production-ready IR systems such as web search engines. Several approaches were proposed to reduce the model size and the inference computation time in transformer networks [59]. Most of them focus on the compression of the neural network to reduce their complexity and, consequently, to reduce their inference time.

Neural network *pruning* consists of removing weights and activation functions in a neural network to reduce the memory needed to store the network parameters. The objective of pruning is to convert the weight matrix of a dense neural network to a sparse structure, which can be stored and processed more efficiently. Pruning techniques work both at learning time and as a post-learning step. In the first category, Pan et al. propose regularization techniques focused at removing redundant neurons at training time [145]. Alternatively, in the second category, Han et al. propose to remove the smallest weights in terms of magnitude and their associated edges to shrink the size of the network [58].

Another research line focuses on improving the efficiency of a network is weight *quantization*. The techniques in this area aim at reducing the number of bits necessary to represent the model weights: from the 32 bits necessary to represent a float to only a few bits [74]. The state of the art network quantization techniques [5, 216] aims at

quantizing the network weights using just 2-3 bits per parameter. These approaches proved effective on convolutional and recurrent neural networks.

A third research line employed to speed-up neural networks is *knowledge distillation* [67]. It aims to transform the knowledge embedded in a large network (called teacher) into a smaller network (called student). The student network is trained to reproduce the results of the teacher networks using a simpler network structure, with less parameters than those used in the teacher network. Several strategies have been proposed to distill knowledge in pretrained transformer networks such as BERT [79, 191, 192].

Within the realm of transformer-based models for ad-hoc ranking, to my knowledge only a few works have acknowledged that retrieval speed is substantially impacted by using a deep transformer network [69, 136, 138]. Nogueira et al. use a transformer-based model to expand document terms at index time, but it comes at a great cost to ranking performance: a trade-off that they state can be worthwhile [138]. A recent extension to that work [136] suggests that the pretrained transformer itself can have a substantial impact on the effectiveness. Hofstätter et al. [69] recently demonstrated that a smaller non-pretrained transformer network can learn to capture content, though the approach does not benefit from the massive amount of pre-training data that contextualized language models offer.

4.2 Pre-computing Representations

In this section, we explore a general technique for reducing the query-time computational cost of trasnformer-based rankers, such as BERT. We exploit a primary characteristic of ad-hoc ranking: an initial indexing phase can be employed to preprocess documents in the collection to improve query-time performance. Specifically,

we observe that much of the term interaction at query time happens locally within either the query or document, and only the last few layers of a deep transformer network are required to produce effective ranking scores once these representations are built. Thus, documents can be processed at index time through part of the network without knowledge of the query. The output of this partial network computation is a sequence of contextualised term representations. These representations can then be stored and used at query time to finish the processing in conjunction with the query. This approach can be trained end-to-end by masking the attention across the query and document during training time (i.e., disallowing the document from attending to the query and vice versa.) We call this approach PreTTR (Precomputing Transformer Term Representations). An overview of this approach is shown in Figure 4.1.

At train time, a transformer network is fine-tuned for ad-hoc document ranking. This transformer network masks attention scores in the first l layers, disallowing interactions between the query and the document. At index time, each document in the collection is processed through the first l layers, and the resulting term representations are stored. At query time, the query is processed through the first l layers, and then combined with the document term representations to finish the ranking score calculation.

Since term representations of each layer can be large (e.g., 768 float values per document term in the base version of BERT), we also propose a compression approach. This approach involves training an encoding layer between two transformer layers that produces representations that can replicate the attention patterns exhibited by the original model. We experimentally show that all these processes result in a much faster network at query time, while having only a minimal impact on the ranking performance and a reasonable change in index size. The settings of PreTTR (amount of pre-computation, degree of compression) can be adjusted depending on the needs

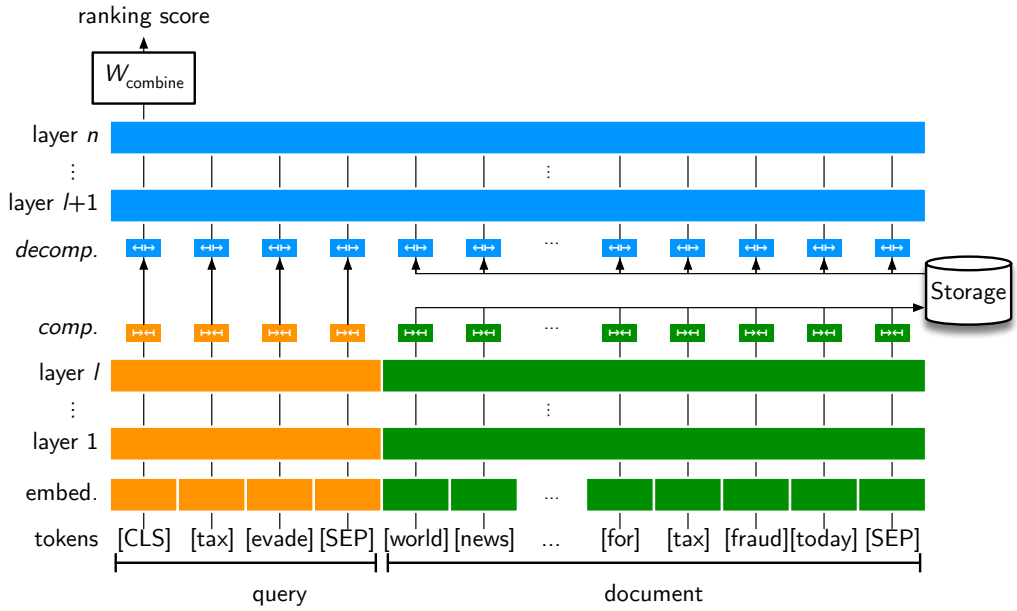

Figure 4.1 Overview of PreTTR. Compressed term representations for document layers 1 to l are computed and stored at index time (green segments) while term representations for query layers 1 to l (orange segments) and joint query-document representations for layers $l+1$ to n (blue segments) are computed at query time to produce the final ranking score. Compression and decompression can optionally be applied between layers l and $l+1$ to reduce the storage needed for the document term representations.

of the application. These are all critical findings that are required to allow transformer networks to be used in practical search environments. Specifically, the lower computation overhead reduces query-time latency of using transformer networks for ranking, all while still yielding the substantial improvements to ranking accuracy that transformer-based rankers offer.

Table 4.1 Table of symbols for PreTTR.

Symbol(s)	Definition
$\mathbf{q}$	Query
$\mathbf{d}$	Document
$R(\mathbf{q}, \mathbf{d})$	Neural ranking architecture
$T(\mathbf{s})$	Transformer network
$\mathbf{s}$	a sequence of input tokens
E	Embedding layer
L_i	Transformer encoding layer
$\mathbf{s}_i$	Transformer token representations after layer i
$\mathbf{a}_i$	Attention weights used in layer i
c	Classification representation
d	Dimension of the classification representation
m	Length of sequence $\mathbf{s}$
h	Number of attention heads per layer
n	Number of layers in T
$W_{combine}$	Vanilla BERT weight combination
l	Layer number the transformer is executed for precomputing document term vectors
e	Compressed size
$\mathbf{r}$	Compressed representation after layer l
W_{comp}, $\mathbf{b}_{comp}$	Compression parameters
W_{decomp}, $\mathbf{b}_{decomp}$	De-compression parameters
$\hat{\mathbf{s}}_l$	De-compressed representation after layer l

4.2.1 Methodology

Preliminaries

We now introduce the notation used in this section (see Table 4.1 for a summary of the notation). Let a generic transformer network $T : \mathbf{s} \mapsto c$ map a sequence $\mathbf{s}$ of m tokens (e.g., query and document terms) to a d-dimensional output representation $c \in \mathbb{R}^d$. As depicted in Figure 4.1, the transformer network is composed by an initial embedding layer E and by n layers $L_1, \ldots, L_n$. The embedding layer E maps each of the m input tokens into the initial d-dimensional token representations matrix $\mathbf{s}_0 \in \mathbb{R}^{m \times d}$. Each layer L_i takes the token representations matrix $\mathbf{s}_{i-1} \in R^{m \times d}$ from the previous layer L_{i-1} and produces a new representations matrix $\mathbf{s}_i \in \mathbb{R}^{m \times d}$. The spe-

cific representation used and operations performed in E and L_i depend on the specific transformer architecture (e.g., BERT uses token, segment, and position embeddings for the embedding layer E and self-attention, a feed-forward layer, and batch normalization in each layer L_i). However, the primary and common component of each layer L_i is the self-attention mechanism and associated procedure. When the transformer network is trained, every layer produces a self-attention tensor $\mathbf{a}_i \in \mathbb{R}^{h \times m \times m}$, where h is the number of attention heads per layer, i.e., the number of attention "representation subspaces" per layer. A general description of this process is given by Vaswani et al. [198], while different transformer architectures may have tweaks to this general structure or pre-training procedure.

We assume a special output classification token, e.g., `[CLS]` in BERT, is included as a token in c, and that the final representation of this token is used as the final output of the transformer network, i.e., $c = T(\mathbf{s})$. Without loss of generality, here we only concern ourselves with the `[CLS]` output classification token, i.e., we ignore other token representation outputs; this is the special token representation that models such as BERT use to generate ranking scores.

We illustrate how neural transformer networks are used in a ranking scenario. We follow and generalize the Vanilla BERT model from Section 3.2. Let a ranking function $R(\mathbf{q}, \mathbf{d}) \in \mathbb{R}$ map a query $\mathbf{q}$ and a document $\mathbf{d}$ to a real-valued ranking score. Neural rankers based on transformer networks such as Vanilla BERT compute the ranking score by feeding the query-document pair into the transformer. Given a query $\mathbf{q}$ and a document $\mathbf{d}$, their tokens are concatenated into a suitable transformer input, e.g., $\mathbf{s} = [\texttt{CLS}]; \mathbf{q}; [\texttt{SEP}]; \mathbf{d}; [\texttt{SEP}]$, where ";" represents the concatenation operator.[1] The output of the transformer network corresponding to this input is then linearly

[1]We use the BERT convention of `[CLS]` and `[SEP]` to represent the classification and separation tokens, respectively.

combined using a tuned weight matrix $W_{combine} \in \mathbb{R}^{d \times 1}$ to compute the final ranking score as follows:

$$R(\mathbf{q}, \mathbf{d}) = T\big([\texttt{CLS}]; \mathbf{q}; [\texttt{SEP}]; \mathbf{d}; [\texttt{SEP}]\big) W_{combine}. \tag{4.1}$$

The processing time of state-of-the-art neural rankers based on transformer networks is very high, e.g., approximately 50 documents ranked per second on a modern GPU, making such rankers impractical for most ad-hoc retrieval tasks.

To gain an understanding of where are the most expensive components of a transformer network such as the Vanilla BERT model, we measure the run-times of the main steps of the model. We find that most of the processing is performed in the computations involving the transformer's layers. In particular, about 50% of the total time is spent performing attention-related tasks. Moreover, the feed-forward step of the transformer (consisting of intermediate and output in diagram) accounts for about 48% of the total time, and is largely due to the large intermediate hidden representation size for each token. This breakdown motivates the investigation of possible solutions to reduce the processing time of transformer networks, in particular in reducing the time spent in traversing the transformer's layers.

PreTTR: Precomputing Transformer Term Representations

We improve the query time performance of transformer models by precomputing document term representations partially through the transformer network (up to transformer layer l). We then use these representations at query time to complete the execution of the network when the query is known.

This is accomplished at model training time by applying an attention mask to layers $L_1, L_2, \ldots, L_l$, in which terms from the query are not permitted to attend to terms from the document and vice versa. In layers $L_{l+1}, \ldots, L_n$, this attention

mask is removed, permitting any token to attend to any other token. Once trained, the model is used at both index and query time. At index time, documents are encoded (including the trailing `[SEP]` token)[2] by the transformer model through layers $L_1, L_2, \ldots, L_l$ without a query present (Figure 4.1, green segments). The token representations generated at index time at layer L_l are then stored to be reused at query time (Figure 4.1, document storage between layers L_l and L_{l+1}). To answer a query, candidate documents are selected, e.g., the top documents retrieved by a first-stage simple ranking model [197], and precomputed term representations are loaded. The query terms (including the leading `[CLS]` and training `[SEP]` tokens) are encoded up to layer L_l without a document present (Figure 4.1, orange segments). Then, the representations from the query and the document are joined, and the remainder of the transformer network is executed over the entire sequence to produce a ranking score (Figure 4.1, blue segments).

Since (1) the length of a query is typically much shorter than the length of a document, (2) the query representations can be re-used for each document being ranked, (3) each transformer layer takes about the same amount of time to execute, and (4) the time needed to perform term embedding is comparatively low, PreTTR decreases by about $\frac{n-l}{n}$ the cost of traversing the transformer network layers. With a sufficiently large value of l, this results in considerable time savings. Note that this reduction can be at most equal to $\frac{1}{n}$ because, when $l = n$, no information about the document ever contributes to the ranking score, resulting in identical scores for every document. Moreover, we show experimentally that this can be further improved by limiting the computation of the final layer to only the `[CLS]` representation.

[2]There is evidence that the separator token performs an important function for pretrained transformer models, by acting as a no-op for the self-attention mechanism [23].

Token Representation Compression

Although PreTTR can reduce the run-time cost of traversing the first l layers of the transformer network at query time, the solution proposed might be costly in terms of storage requirements because the representation size d is quite large (e.g., 1024, 768 or 512 float values per token). To address this issue, we propose a new token compression technique that involves pre-training a simple encoder-decoder network. This network is able to considerably reduce the token representation size. We opt for this approach because it can fit seamlessly into the transformer network, while reducing the number of dimensions needed to represent each token. The compressor is added as an additional component of the transformer network between layers L_l and L_{l+1}. We compress the input by using a simple feed-forward and normalization procedure, identical to the one used within a BERT layer to transform the output (but with a *smaller* internal representation rather than a larger one). We optimize the weights for the compression network in two stages: (1) an initial pre-training stage on unlabeled data, and (2) a fine-tuning stage when optimizing for relevance.

For a compressed size of e values, a two-step procedure is used. First, the compressed representations $\mathbf{r} \in \mathbb{R}^{m \times e}$ are built using $\mathbf{r} = \text{GELU}(\mathbf{s}_l W_{comp} + \mathbf{b}_{comp})$, where $\text{GELU}(\cdot)$ is a Gaussian Error Linear Unit [66], and $W_{comp} \in \mathbb{R}^{d \times e}$ and $\mathbf{b}_{comp} \in \mathbb{R}^e$ are the new learned weight parameters. These compressed representations $\mathbf{r}$ can be stored in place of $\mathbf{s}_l$. Second, the compressed representations $\mathbf{r}$ are then expanded back out to $\hat{\mathbf{s}}_l \in \mathbb{R}^{m \times d}$ via a second linear transformation involving the learned weight parameters W_{decomp}, $\mathbf{b}_{decomp}$, and batch normalization. The decompressed representations $\hat{\mathbf{s}}_l$ are then used in place of the original representation $\mathbf{s}_l$ for the remaining layers of the transformer.

In preliminary experiments, we found the compression and decompression parameters to be difficult to learn jointly with the ranker itself. Thus, we instead propose a pre-training approach to provide an effective initialization of these parameters. We want the transformer network with the compression mechanism to behave similarly to that of the network without such compression: we do not necessarily care about the exact representations themselves. Thus, we use an attention-based loss function. More specifically, we optimize our compression/decompression network to reduce the mean squared error of the attention scores in the last $n - l$ layers of the compressed transformer network and the original transformer network. Thus, the loss function we use to train our compression and decompression network is:

$$\mathcal{L}(\mathbf{a}_{l+1}, \ldots, \mathbf{a}_n, \hat{\mathbf{a}}_{l+1}, \ldots, \hat{\mathbf{a}}_n) = \frac{1}{n-l} \sum_{i=l+1}^{n} \text{MSE}(\mathbf{a}_i, \hat{\mathbf{a}}_i), \tag{4.2}$$

where a_i represents the attention scores at layer i from the unmodified transformer network, $\hat{a}_i$ represents the attention scores at layer i from the transformer network with the compression unit, and $\text{MSE}(\cdot)$ is the mean squared error function. With this loss function, the weights can be pre-trained on a massive amount of unlabeled text. We use this procedure as an initial pre-training step; we further fine-tune the weights when optimizing the entire ranking network for relevance.

4.2.2 Experiment

To test the effectivness of PreTTR, we use the Vanilla transformer model introduced in Section 3.2. This model yields comparable performance to other leading formulations, while being simpler, e.g., no paragraph segmentation required, as is needed by `FirstP`/`MaxP`/`SumP` [36], or alternative training datasets and sentence segmentation, as required by the system of Yang et al. [220]. Vanilla BERT encodes as much of the document as possible (adhering to the transformer maximum input

length constraint), and averages the classification embeddings when multiple document segments are required. We employ the same optimal hyper-parameters for the model presented in [110]. For our primary experiments, we use the pretrained `bert-base-uncased` [44]. We do not test with the `large` variants of BERT because the larger model exhibits only marginal gains for ranking tasks, while being considerably more expensive to run [135]. To show the generality of our approach we present tests conducted also for other pretrained transformers: a version of BERT that was more effectively pre-trained, i.e., RoBERTa [105] (`roberta-base`) and a smaller (distilled) version of BERT, i.e., DistilBERT [177] (`distilbert-base-uncased`).

We test our models using the TREC WEbTrack 2012 and TREC Robust 2004 datasets, introduced in Section 2.1.

Training

We train all transformer models using pairwise softmax loss [43] and the Adam optimizer [86] with a learning rate of 2×10^{-5}. We employ a batch size of 16 pairs of relevant and non-relevant documents with gradient accumulation. Training pairs are selected randomly from the top-ranked documents in the training set, where documents that are labeled as relevant are treated as positive, and other top-ranked documents are considered negative. Every 32 batches, the model is validated, and the model yielding the highest performance on the validation set is selected for final evaluation.

For training the document term compressor/decompressor (as described in Section 4.2.1), we use the Wikipedia text from the TREC Complex Answer Retrieval (CAR) dataset [47] (version 2.0 release). This dataset was chosen because it overlaps with the data on which BERT was originally trained on, i.e., Wikipedia, and was used both for evaluation of passage ranking approaches [129] and as a weak supervision

dataset for training neural models [112]. We sample text pairs using combinations of headings and paragraphs. Half the pairs use the heading associated with the paragraph, and the other half use a random heading from a different article, akin to the next sentence classification used in BERT pre-training. The compression and decompression parameters (W_{comp}, $\mathbf{b}_{comp}$, W_{decomp}, and $\mathbf{b}_{decomp}$) are trained to minimize the difference in attention scores, as formulated in Eq. (4.2). We found that the compressor training process converged by $2M$ samples.

Evaluation

Since the transformer network is employed as a final-stage re-ranker, we evaluate the performance of our approach on each dataset using two precision-oriented metrics. Our primary metric for both datasets is P@20 (also used for model validation). Following the evaluation convention from prior work [110], we use ERR@20 for TREC WebTrack 2012 and nDCG@20 for TREC Robust 2004 as secondary metrics.

We also evaluate the query-time latency of the models. We conduct these experiments using commodity hardware: one GeForce GTX 1080 Ti GPU. To control for factors such as disk latency, we assume the model and term representations are already loaded in the main memory. In other words, we focus on the impact of the model computation itself. However, the time spent moving the data to and from the GPU memory is included in the time.

Baselines

The focus of this work is to reduce the query-time latency of using Vanilla transformer models, which are among the state-of-the-art neural ranking approaches. Thus, our primary baseline is the unmodified Vanilla transformer network. To put the results in context, we also include the BM25 results tuned on the same training data. We

tune BM25 using grid search with Anserini's implementation [217], over k_1 in the range of 0.1–4.0 (by 0.1) and b in the range of 0.1–1.0 (by 0.1). We also report results for CEDR-KNRM from Section 3.2, which outperform the Vanilla transformer approaches. However, it come with its own query-time challenges. Specifically, since it uses the term representations from every layer of the transformer, this would require considerably more storage. To keep our focus on the typical approach, i.e., using the `[CLS]` representation for ranking, we leave it to future work to investigate ways in which to optimize the CEDR model.[3] We also report results for Birch [3], which exploits transfer learning from the TREC Microblog dataset. To keep the focus of this work on the effect of pre-computation, we opt to evaluate in the single-domain setting.

Results

We report the results of a comprehensive experimental evaluation of the proposed PreTTR approach. In particular, we aim at investigating the following research questions:

RQ1 What is the impact of PreTTR on the effectiveness of the Vanilla BERT transformer network in ad-hoc ranking?

RQ2 What is the impact of the token representation compression on the effectiveness of PreTTR?

RQ3 What is the impact of the proposed PreTTR approach on the efficiency of Vanilla BERT when deployed as a second stage re-ranker?

[3]We note that techniques such as LSH hashing can reduce the storage requirements for CEDR, as it uses the representations to compute query-document similarity matrices, as demonstrated by [78].

Table 4.2 Breakdown of ranking performance when using a PreTTR-based Vanilla BERT ranking.

	WebTrack 2012		Robust 2004	
Ranker	P@20	ERR@20	P@20	nDCG@20
Base	**0.3460**	0.2767	0.3784	0.4357
$l = 1$	0.3270	**0.2831**	0.3851	**0.4401**
$l = 2$	0.3170	0.2497	0.3821	0.4374
$l = 3$	0.3440	0.2268	**0.3859**	0.4386
$l = 4$	0.3280	0.2399	0.3701	0.4212
$l = 5$	0.3180	0.2170	0.3731	0.4214
$l = 6$	0.3270	0.2563	0.3663	0.4156
$l = 7$	0.3180	0.2255	0.3656	0.4139
$l = 8$	0.3140	0.2344	0.3636	$\downarrow$ 0.4123
$l = 9$	0.3130	0.2297	0.3644	$\downarrow$ 0.4106
$l = 10$	0.3360	0.2295	0.3579	$\downarrow$ 0.4039
$l = 11$	0.3380	$\downarrow$ 0.1940	$\downarrow$ 0.2534	$\downarrow$ 0.2590
Tuned BM25	0.2370	0.1418	0.3123	0.4140
Vanilla BERT [110]	-	-	0.4042	0.4541
CEDR-KNRM [110]	-	-	0.4667	0.5381
Birch [3]	-	-	0.4669	0.5325

Query and document encodings are jointed at layer l. Statistically significant differences with the base model are indicated by $\downarrow$ (paired t-test by query, $p < 0.01$).

RQ4 What is the impact of PreTTR when applied to first $n-1$ layers of a transformer network?

RQ5 What is the impact of PreTTR when applied to different transformer networks such as RoBERTA and DistilBERT?

To answer RQ1 we first evaluate the effect of the precomputation of term representations. Table 4.2 provides a summary of the ranking performance of PreTTR-based

Vanilla BERT at layer l. At lower values of l, the ranking effectiveness remains relatively stable, despite some minor fluctuations. We note that these fluctuations are not statistically significant when compared with the base model (paired t-test, 99% confidence interval) and remain considerably higher than the tuned BM25 model. We also tested using a two one-sided equivalence (TOST) and found similar trends (i.e., typically the the significant differences did not exhibit significant equivalence.) In the case of TREC WebTrack 2012, the model achieves comparable P@20 performance w.r.t. the base model with only a single transformer layer (12), while the first 11 layers are precomputed. Interestingly, the ERR@20 suffers more than P@20 as more layers are precomputed. This suggests that the model is able to identify generally-relevant documents very effectively with only a few transformer layers, but more are required to be able to identify the subtleties that contribute to greater or lesser degrees of relevance. Although it would ideally be best to have comparable ERR@20 performance in addition to P@20, the substantial improvements that this approach offers in terms of query-time latency may make the trade-off worth it, depending on the needs of the application.

On the TREC Robust 2004 newswire collection, precomputing the first 10 layers yields comparable P@20 performance w.r.t. the base model. Interestingly, although $l = 11$ yields a relatively effective model for WebTrack, Robust performance significantly suffers in this setting, falling well below the BM25 baseline. We also observe a significant drop in nDCG@20 performance at $l = 8$, while P@20 performance remains stable until $l = 11$. This is similar to the behavior observed on WebTrack: as more layers are precomputed, the model has a more difficult time distinguishing graded relevance.

Table 4.3 Ranking performance at various compression sizes.

TREC WebTrack 2012

Compression	$l = 7$	$l = 8$	$l = 9$	$l = 10$	$l = 11$
			P@20		
(none)	0.3180	0.3140	0.3130	**0.3360**	**0.3380**
$e = 384$ (50%)	**0.3430**	**0.3260**	0.2980	**0.3360**	0.3090
$e = 256$ (67%)	0.3380	0.3120	↑ **0.3440**	0.3260	0.3250
$e = 128$ (83%)	0.3100	0.3210	0.3320	0.3220	0.3370
			ERR@20		
(none)	0.2255	**0.2344**	0.2297	**0.2295**	0.1940
$e = 384$ (50%)	0.2086	0.2338	0.1685	0.2233	**0.2231**
$e = 256$ (67%)	↑ **0.2716**	0.2034	↑ **0.2918**	0.1909	0.2189
$e = 128$ (83%)	0.2114	0.2234	0.2519	0.2239	0.2130

TREC Robust 2004

Compression	$l = 7$	$l = 8$	$l = 9$	$l = 10$	* $l = 11$
			P@20		
(none)	**0.3656**	**0.3636**	**0.3644**	**0.3579**	0.2534
$e = 384$ (50%)	0.3587	↓ 0.3369	↓ 0.3435	0.3522	**0.2687**
$e = 256$ (67%)	↓ 0.2950	0.3623	↓ 0.2695	0.3535	0.2635
$e = 128$ (83%)	↓ 0.2461	↓ 0.2530	↓ 0.2499	↓ 0.2607	0.2655
			nDCG@20		
(none)	**0.4139**	**0.4123**	**0.4106**	**0.4039**	0.2590
$e = 384$ (50%)	0.4098	↓ 0.3720	↓ 0.3812	0.3895	↑ **0.2807**
$e = 256$ (67%)	↓ 0.3130	0.4074	↓ 0.2753	0.3983	0.2694
$e = 128$ (83%)	↓ 0.2454	↓ 0.2568	↓ 0.2533	↓ 0.2608	0.2713

Statistically significant increases and decreases in ranking performance (compared to the model without compression) are indicated with ↑ and ↓, respectively (paired t-test by query, $p < 0.01$).

We observe that the highest-performing models (metric in bold) are not always the base model. However, we note that these scores do not exhibit statistically significant differences when compared to the base model.

In summary, we answer RQ1 by showing that Vanilla BERT can be successfully trained by limiting the interaction between query terms and document terms, and that this can have only a minimal impact on ranking effectiveness, particularly in terms in the precision of top-ranked documents. This is an important result because it shows that document term representations can be built independently of the query at index time.

To answer RQ2, we run the Vanilla BERT model with varying sizes e of the compressed embedding representations over the combination layers l that give the most benefit to query latency time (i.e., $l = 7, 8, 9, 10, 11$). Layers $l \leq 6$ are not considered because they provide less computational benefit (taking about one second or more per 100 documents. See Table 4.3 for a summary of the results on TREC WebTrack 2012 and Robust 2004. We find that the representations can usually be compressed down to at least $e = 256$ (67% of the original dimension of 768) without substantial loss in ranking effectiveness. In Robust, we observe a sharp drop in performance at $e = 128$ (83% dimension compression) at layers 7–10. There is no clear pattern for which compression size is most effective for WebTrack 2012. Note that these differences are generally not statistically significant. This table shows that, to a point, there is a trade-off between the size of the stored representations and the effectiveness of the ranker.

Without any intervention, approximately 112TB of storage would be required to store the full term vectors for ClueWeb09-B (the document collection for TREC WebTrack 2012). For web collections, this can be substantially reduced by eliminating undesirable pages, such as spam. Using recommended settings for the spam filtering

approach proposed by Cormack et al. [31] for ClueWeb09-B, the size can be reduced to about 34TB. Using our compression/decompression approach, the storage needed can be further reduced, depending on the trade-off of storage, query-time latency, and storage requirements. If using a dimension $e = 128$ for the compressed representation (with no statistically significant differences in effectiveness on WebTrack), the size is further reduced to 5.7TB, which yields a 95% of space reduction. We also observed that there is little performance impact by using 16-bit floating point representations, which further reduces the space to about 2.8TB. Although this is still a tall order, it is only about 2.5% of the original size, and in the realm of reasonable possibilities. We leave it to future work to investigate further compression techniques, such as kernel density estimation-based quantization [184].

Since the size scales with the number of documents, the storage requirements are far less for smaller document collections such as newswire. Document representations for the TREC Disks 4 & 5 (the document collection for the Robust 2004) can be stored in about 195GB, without any filtering and using the more effective $e = 256$ for the dimension of the compressed representation.

In summary, regarding RQ2, we show that, through our compression technique, one can reduce the storage requirements of PreTTR. With a well-trained compression and decompression weights, this can have minimal impact on ranking effectiveness.

The reduction of the re-ranking latency achieved by our proposed PreTTR is considerable. To answer RQ3, in Table 4.4 we report an analysis of the re-ranking latency of PreTTR-based Vanilla BERT when precomputing the token representations at a specific layer l and a comparison against the base model, i.e., Vanilla BERT. Without our approach, re-ranking the top 100 results for a query using Vanilla BERT takes around 2 seconds. Instead, when using PreTTR-based Vanilla BERT at layer $l = 11$, which yields comparable P@20 performance to the base model on the TREC Web-

Table 4.4 Vanilla BERT query-time latency measurements for re-ranking the top 100 documents on TREC WebTrack 2012 and TREC Robust 2004.

	TREC WebTrack 2012					Robust04
Ranker	Total	Speedup	Query	Decom.	Combine	Total
Base	1.941s	(1.0×)	-	-	-	2.437s
$l = 1$	1.768s	(1.1×)	**2ms**	10ms	1.756s	2.222s
$l = 2$	1.598s	(1.2×)	3ms	10ms	1.585s	2.008s
$l = 3$	1.423s	(1.4×)	5ms	10ms	1.409s	1.792s
$l = 4$	1.253s	(1.5×)	6ms	10ms	1.238s	1.575s
$l = 5$	1.080s	(1.8×)	7ms	10ms	1.063s	1.356s
$l = 6$	0.906s	(2.1×)	9ms	10ms	0.887s	1.138s
$l = 7$	0.735s	(2.6×)	10ms	10ms	0.715s	0.922s
$l = 8$	0.562s	(3.5×)	11ms	10ms	0.541s	0.704s
$l = 9$	0.391s	(5.0×)	12ms	10ms	0.368s	0.479s
$l = 10$	0.218s	(8.9×)	14ms	10ms	0.194s	0.266s
$l = 11$	**0.046s**	**(42.2×)**	15ms	10ms	**0.021s**	**0.053s**

The latency is broken down into time to compute query representations up through layer l, the time to decompress document term representations, and the time to combine the query and document representations from layer $l + 1$ to layer n. The $l = 11$ setting yields a 42× speedup for TREC WebTrack, while not significantly reducing the ranking performance.

Track 2012 collection, the re-ranking process takes 46 milliseconds for 100 documents, i.e., we achieve a 42.0× speedup. One reason this performance is achievable is because the final layer of the transformer network does not need to compute the representations for each token; only the representations for the `[CLS]` token are needed, since it is the only token used to compute the final ranking score. Thus, the calculation of a full self-attention matrix is not required. Since the `[CLS]` representation is built in conjunction with the query, it alone can contain a summary of the query terms. Furthermore, since the query representation in the first l layers is independent of the document, these representations are re-used among all the documents that are re-ranked. Of the time spent during re-ranking for $l = 11$, 32% of the time is spent building the query term representation, 21% of the time is spent decompressing the document term representations, and the remainder of the time is spent combining the query and document representations. Moreover, when using PreTTR-based Vanilla BERT at layer $l = 10$, the transformer network needs to perform a round of computations on all the term representations. Nevertheless, in this case, our PreTTR approach leads to a substantial speedup of 8.9× w.r.t. Vanilla BERT. We also observe that the time to decompress the term representations (with $e = 256$) remains a constant overhead, as expected. We observe a similar trend when timing the performance of Robust 2004, though we would recommend using $l \leq 10$ for this dataset, as $l = 11$ performs poorly in terms of ranking effectiveness. Nonetheless, at $l = 10$, Robust achieves a 9.2× speedup, as compared to the full model.

In summary, regarding RQ3, we show that the PreTTR approach can save a considerable amount of time at query-time, as compared to the full Vanilla BERT model. These time savings can make it practical to run transformer-based rankers in a real-time query environment.

We answer RQ4 by highlighting a first interesting difference between the WebTrack and the Robust ranking performance: the effectiveness at $l = 11$ (Table 4.2). For WebTrack, the performance is comparable in terms of P@20, but suffers in terms of ERR@20. For Robust, the performance suffers drastically. We attribute this to differences in the dataset characteristics. First, let us consider what happens in the $l = 11$ case. Since it is the final layer and only the representation of the `[CLS]` token is used for ranking, the only attention comparisons that matter are between the `[CLS]` token and every other token (not a full comparison between every pair of tokens, as is done in other layers). Thus, a representation of the entire query must be stored in the `[CLS]` representation from layer 11 to provide an effective comparison with the remainder of the document, which will have no contribution from the query. Furthermore, document token representations will need to have their context be fully captured in a way that is effective for the matching of the `[CLS]` representation. Interestingly, this setting blurs the line between representation-focused and interaction-focused neural models.

Now we will consider the characteristics of each dataset. The queries in the TREC WebTrack 2012 are typically shorter (mean: 2.0, median: 2, stdev: 0.8) than those from Robust (mean: 2.7, median: 3, stdev: 0.7). This results in queries that are more qualified, and may be more difficult to successfully represent in a single vector.

To answer RQ4, we observe that the ranking effectiveness when combining with only a single transformer layer can vary depending on dataset characteristics. We find that in web collections (an environment where query-time latency is very important), it may be practical to use PreTTR in this way while maintaining high precision of the top-ranked documents.

Numerous pre-trained transformer architectures exist. We now answer RQ5 by showing that PreTTR is not only effective on BERT, but its ability of reducing

Table 4.5 WebTrack 2012 using two other Vanilla transformer architectures: RoBERTa and DistilBERT.

	RoBERTA [105]		DistilBERT [177]	
Ranker	P@20	ERR@20	P@20	ERR@20
Base	0.3370	0.2609	0.3110	0.2293
$l = 1$	0.3380	**0.2796**	0.3220	0.1989
$l = 2$	0.3370	0.2207	0.3340	**0.2771**
$l = 3$	0.3530	0.2669	0.3070	0.1946
$l = 4$	**0.3620**	0.2647	0.3350	0.2281
$l = 5$	0.2950	0.1707	**0.3350**	0.2074
$l = 6$	0.3000	0.1928	-	-
$l = 7$	0.3350	0.2130	-	-
$l = 8$	0.3220	0.2460	-	-
$l = 9$	0.3180	0.2256	-	-
$l = 10$	0.3140	0.1603	-	-
$l = 11$	0.3210	0.2241	-	-

Note that DistilBERT only has 6 layers; thus we only evaluate $l \in [1, 5]$ for this model. There are no statistically significant differences between the Base Model and any of the PreTTR variants (paired t-test, $p < 0.01$).

ranking latency by preserving quality holds also on other transformer variants. We investigate both the popular RoBERTa [105] model and the DistilBERT [177] model. These represent a model that uses a more effective pre-training process, and a smaller network size (via model distillation), respectively. Results for this experiment are shown in Table 4.5. We first observe that the unmodified RoBERTa model performs comparably with the BERT model, while the DistilBERT model performs slightly worse. This suggests that model distillation alone may not be a suitable solution to address the poor query-time ranking latency of transformer networks. With each value of l, we observe similar behavior to BERT: P@20 remains relatively stable, while ERR@20 tends to degrade. Interestingly, at $l = 2$ DistilBERT's ERR@20 performance peaks at 0.2771. However, this difference is not statistically significant, and thus we cannot assume it is not due to noise.

We also tested the query-time latency of RoBERTa and DistilBERT. With 12 layers and a similar neural architecture, RoBERTa exhibited similar speedups as BERT, with up to a 56.3× speedup at $l = 11$ (0.041s per 100 documents, down from 1.89s). With only 6 layers, the base DistilBERT model was faster (0.937s), and was able to achieve a speedup of 24.1× with $l = 5$ (0.035s).

In summary, we show that the PreTTR approach can be successfully generalized to other transformer networks (RQ5). We observed similar trends to those we observed with BERT in two transformer variants, both in terms of ranking effectiveness and efficiency.

4.2.3 Summary

As was shown in Chapter 3, transformer networks, such as BERT, present a considerable opportunity to improve ranking effectiveness. However, relatively little attention has been paid to the effect that these approaches have on query execution time.

We showed that these networks can be trained in a way that is more suitable for query-time latency demands. Specifically, we showed that query execution time can be improved by up to 42× for web document ranking, with minimal impact on P@20. Although this approach requires storing term representations for documents in the collection, we proposed an approach to reduce this storage required by 97.5% by pre-training a compression/decompression function and using reduced-precision (16 bits) floating point arithmetic. We experimentally showed that the approach works across transformer architectures, and we demonstrated its effectiveness on both web and news search. These findings are particularly important for large-scale search settings, such as web search, where query-time latency is critical. These findings validate Hypothesis 2.1.

4.3 Learning Efficient Sparse Representations for Ranking

Armed with the knowledge that document representations can be precomputed at index time, we set out to build a ranking architecture that attempts to minimize the cost at query time as much as possible. To this end, we propose a new ranking architecture that performs modeling of term importance (i.e., salience) and expansion over a contextualized language model to build query and document representations. We call this approach EPIC (Expansion via Prediction of Importance with Contextualization). At query time, EPIC can be employed as an inexpensive re-ranking method because document representations can be pre-computed at index time. EPIC improves upon the prior state of the art on the MS-MARCO passage ranking dataset by substantially narrowing the effectiveness gap between practical approaches with subsecond retrieval times and those that are considerably more expensive, e.g., those using BERT as a re-ranker. Furthermore, the proposed representations are inter-

pretable because the dimensions of the representation directly correspond to the terms in the lexicon. An overview is shown in Figure 4.2.

4.3.1 Methodology

Overview and notation. Our model follows the representation-focused neural ranking paradigm. That is, we train a model to generate a query and document (or passage)[4] representation in a given fixed-length vector space, and produce a ranking score by computing a similarity score between the two representations.

Assume that queries and documents are composed by sequences of terms taken from a vocabulary V. Any sequence of terms, either a query or a document, is firstly represented as a sequence of vectors using a contextualized language model like BERT [44]. More formally, let $f : V^n \to \mathbb{R}^{n \times e}$ denote such a function associating an input sequence s of n terms $t_1, \ldots, t_n$ to their n embeddings $f_1(s), \ldots, f_n(s)$, where $f_i(s) \in \mathbb{R}^e$ and e is the size of the embedding. So, a n-term query q is represented with the n embeddings $f_1(q), \ldots, f_n(q)$, and a m-term document d is represented with m embeddings $f_1(d), \ldots, f_m(d)$. Given the embeddings for queries and documents, we

[4]For ease of notation, we refer to passages as documents.

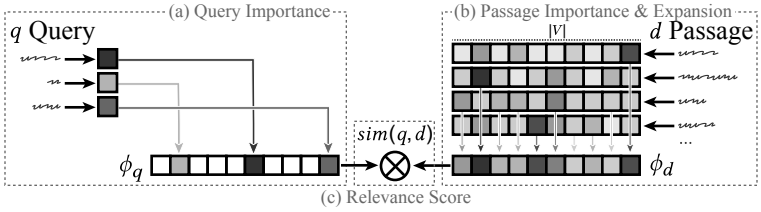

Figure 4.2 Overview of EPIC.

now illustrate the process for constructing query representations, document representations, the final query-document similarity score.

Query representation. A query q is represented as a *sparse* vector $\phi_q \in \mathbb{R}^{|V|}$ (Figure 4.2 (a)). The elements of ϕ_q that correspond to to terms not in the query are set to 0. For each term t_i appearing in the $t_1, \ldots, t_n$ terms of the query q, the corresponding element $\phi_q(t_i)$ is equal to the importance $w_q(t_i)$ of the term w.r.t. the query

$$w_q(t_i) = \ln\Big(1 + \text{softplus}\big(\theta_1^\top f_i(q)\big)\Big), \tag{4.3}$$

where $\theta_1 \in \mathbb{R}^e$ is a vector of learned parameters. The softplus$(\cdot)$ function is defined as softplus$(x) = \ln(1 + e^x)$. The use of softplus ensures that no terms have a negative importance score, while imposing no upper bound. The logarithm prevents individual terms from dominating. When a term appears more than once in a query, the corresponding value of ϕ_q sums up all contributions. The elements of the query representation encode the importance of the terms w.r.t. the query. This approach allows the query representation model to learn to assign higher weights to the query terms that are most important to match given the textual context. Note that the number of elements in the representation is equal to the number of query terms; thus the query processing time is proportional to the number of query terms [197].

Document representation. A document d is represented as a *dense* vector $\phi_d \in \mathbb{R}^{|V|}$ (Figure 4.2 (b)). Firstly, to perform document expansion, each e-dimensional term embedding $f_j(d)$ is projected into a $|V|$-dimensional vector space, i.e., $\psi_j : f_j(d) \mapsto \Theta_2 f_j(d)$, where $\Theta_2 \in \mathbb{R}^{|V| \times e}$ is a matrix of learned parameters. Note that $\psi_j \in \mathbb{R}^{|V|}$, and let $\psi_j(\tau)$ denote the entry of this vector corresponding to term $\tau \in V$. Secondly, the importance $w_d(t_j)$ of the terms w.r.t. the document is computed as in

Eq (4.3):

$$w_d(t_j) = \ln\Big(1 + \text{softplus}\big(\theta_3^\top f_j(d)\big)\Big), \tag{4.4}$$

where $\theta_3 \in \mathbb{R}^e$ is a vector of learned parameters. Thirdly, we compute a factor representing the overall quality $c(d)$ of the document

$$c(d) = \text{sigmoid}(\theta_4^\top d_{\texttt{[CLS]}}), \tag{4.5}$$

where $\theta_4 \in \mathbb{R}^e$ is a vector of learned parameters, and $d_{\texttt{[CLS]}} \in \mathbb{R}^e$ is the embedding produced by the contextualized language model's classification mechanism. We find that this factor helps give poor-quality documents lower values overall. The sigmoid$(\cdot)$ function is defined as: $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$. Finally, for each term τ appearing in the vocabulary, the corresponding element of the document representation $\phi_d(\tau)$ is defined as:

$$\phi_d(\tau) = c(d) \max_{t_j \in d} \big(w_d(t_j)\psi_j(\tau)\big). \tag{4.6}$$

This step takes the maximum score for each term in the vocabulary generated by any term in the document. Since they do not rely on the query, these representations can be computed at index time.

Similarity measure. We use the dot product to compute the similarity between the query and document vectors (Figure 4.2 (c)), i.e.,

$$sim(q, d) = \phi_q^\top \phi_d = \sum_{\tau \in V} \phi_q(\tau)\phi_d(\tau). \tag{4.7}$$

4.3.2 Experiment

We conduct experiments using the MS-MARCO passage ranking dataset (full ranking setting).[5] This dataset consists of approximately 1 million natural-language questions gathered from a query log (average length: 7.5 terms, stddev: 3.1), and 8.8 million

[5] `https://microsoft.github.io/msmarco/`

candidate answer passages (avg length: 73.1, stddev: 28.4). The dataset is shallowly-annotated. Annotators were asked to write a natural-language answer to the given question using a set of candidate answers from a commercial search engine. The annotators were asked to indicate which (if any) of the passages contributed to their answers, which are then treated as relevant to the question. This results in 0.7 judgments per query on average (1.1 judgments per query of the 62% that have an answer). Thus, this dataset has a lot of variation in queries, making it suitable for training neural ranking methods. Although this dataset is limited by the method of construction, the performance on these shallow judgments correlate well with those conducted on a deeply-judged subset [34].

Training. We train our model using the official MS-MARCO sequence of training triples (query, relevant passage, presumed non-relevant passage) using cross-entropy loss. We use BERT-base [44] as the contextualized language model, as it was shown to be an effective foundation for various ranking techniques [35, 110, 135]. We set the dimensionality $|V|$ of our representations to the size of the BERT-base word-piece vocabulary (d=30,522). The embedding size is instead $e = 768$. Θ_2 is initialized to the pre-trained masked language model prediction matrix; all other added parameters are randomly initialized. Errors are back-propagated through the entire BERT model with a learning rate of 2×10^{-5} with the Adam optimizer [86]. We train in batches of 16 triples using gradient accumulation, and we evaluate the model on a validation set of 200 random queries from the development set every 512 triples. The optimal training iteration and re-ranking cutoff threshold is selected using this validation set. We roll back to the top-performing model after 20 consecutive iterations (training iteration 42) without improvement to Mean Reciprocal Rank at 10 (MRR@10).

Baselines and Evaluation. We test our approach by re-ranking the results from several first-stage rankers. We report the performance using MRR@10, the official

Table 4.6 Effectiveness and efficiency of EPIC compared to a variety of baselines..

Method	MS-Marco Dev MRR@10	Latency ms/query
Single-Stage Ranking		
BM25 (from Anserini [217])	0.198	21
doc2query [138]	0.218	48
DeepCT-Index [35]	0.243	15
docTTTTTquery [136]	0.277	63
Representation-based Re-Ranking		
EPIC + BM25 (ours)	0.273	106
pruned $r = 2000$	0.273	104
pruned $r = 1000$	*0.272*	*48*
EPIC + docTTTTTquery (ours)	0.304	78
pruned $r = 2000$	0.304	77
pruned $r = 1000$	*0.303*	*68*
Other Re-Ranking		
Duet (v2, ensemble) [126]	0.252	440
BM25 + TK (1 layer) [70]	0.303	445
BM25 + TK (3 layers) [70]	0.314	640
BM25 + BERT (large) [135]	0.365	3,500*

The values in *italics* represent a good trade-off between effectiveness and query latency. The value marked with * was reported by [136].

evaluation metric, on the MS-MARCO passage ranking Dev set. We measure significance using a paired t-test at $p < 0.01$. We compare the performance of our approach with the following baselines:

- BM25 retrieval from a Porter-stemmed Anserini [217] index using default settings.[6]

- DeepCT-Index [35], a model which predicts document term importance scores, and replaces the term frequency values with these importance scores for first-stage retrieval.

- doc2query [138] and docTTTTTquery [136], document expansion approaches which predict additional terms to add to the document via a sequence-to-sequence transformer model. These terms are then indexed and used for retrieval using BM25. The docTTTTTquery model uses a pre-trained T5 model [159].

- Duet [126], a hybrid representation- and interaction-focused model. We include the top Duet variant on the MS-MARCO leaderboard (version 2, ensemble) to compare with another model that utilizes query and document representations.

- TK [70], a contextualized interaction-based model, focused on minimizing query time. We report results from [70] with the optimal re-ranking threshold and measure end-to-end latency on our hardware.

- BERT Large [135], an expensive contextualized language model-based re-ranker. This approach differs from ours in that it models the query and passage jointly at query time, and uses the model's classification mechanism for ranking.

We also measure query latency over the entire retrieval and re-ranking process. The experiments were conducted on commodity hardware equipped with an AMD Ryzen

[6]We observe that the default settings outperform the BM25 results reported elsewhere and on the official leaderboard (e.g., [138]).

3.9GHz processor, 64GiB DDR4 memory, a GeForce GTX 1080ti GPU, and a SSD drive. We report the latency of each method as the average execution time (in milliseconds) of 1000 queries from the Dev set after an initial 1000 queries is used to warm up the cache. First-stage retrieval is conducted with Anserini [217].

Ranking effectiveness. We report the effectiveness of our approach in terms of MMR@10 in Table 4.6. When re-ranking BM25 results, our approach substantially outperforms doc2query and DeepCT-Index. Moreover, it performs comparably to docTTTTTquery (0.273 compared to 0.277, no statistically significant difference). More importantly, we observe that the improvements of our approach and docTTTTTquery are additive as we achieve a MRR@10 of 0.304 when used in combination. This is a statistically significant improvement, and substantially narrows the gap between approaches with low query-time latency and those that trade off latency of effectiveness (e.g., BERT Large).

To test whether EPIC is effective on other passage ranking tasks as well, we test on the TREC CAR passage ranking benchmark [46]. When trained and evaluated on the 2017 dataset (automatic judgments) with BM25, the MRR increases from 0.235 to 0.353. This also outperforms the DeepCT performance reported by [35] of 0.332.

Effect of document representation pruning. For document vectors, we observe that the vast majority of values are very low (approximately 74% have a value of 0.1 or below, see Figure 4.3). This suggests that many of the values can be pruned with little impact on the overall performance. This is desirable because pruning can substantially reduce the storage required for the document representations. To test this, we apply our method keeping only the top r values for each document. We show the effectiveness and efficiency of $r = 2000$ (reduces vocabulary by 93.4%) and $r = 1000$ (96.7%) in Table 4.6. We observe that the vectors can be pruned to $r = 1000$ with virtually no difference in ranking effectiveness (differences

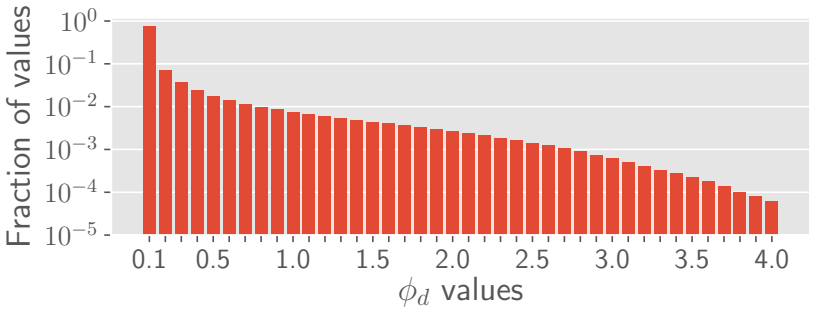

Figure 4.3 Frequencies of EPIC document scores. Note that the values are in log scale. In this figure, values are rounded up to the nearest decimal value.

not statistically significant). We also tested with lower values of r, but found that the effectiveness drops off considerably by $r = 100$ (0.241 and 0.285 for BM25 and docTTTTTquery, respectively).

Ranking efficiency. We find that EPIC can be implemented with a minimal impact on query-time latency. On average, the computation of the query representation takes 18ms on GPU and 51ms on CPU. Since this initial stage retrieval does not use our query representation, it is computed in parallel with the initial retrieval, which reduces the impact on latency. The similarity measure consistently takes approximately 1ms per query (both on CPU and GPU), with the remainder of the time spent retrieving document representations from disk. Interestingly, we observe that the latency of EPIC BM25 is higher than EPIC docTTTTTquery. This is because when re-ranking docTTTTTquery results, a lower re-ranking cutoff threshold is needed than for BM25. This further underscores the importance of using an effective first-stage ranker. When using pruning at $r = 1000$, the computational overhead can be substantially reduced. Specifically, we find that EPIC only adds a 5ms overhead per query to docTTTTTquery, while yielding a significant improvement in effectiveness.

(a) how far does aaa tow , in california
(b) coastal processes are located on what vertebrae
(c) cost of endless pools swim spa

Figure 4.4 Relative EPIC importance scores of sample queries. Darker colors correspond to higher weightsin the query representation.

With pruning at $r = 1000$, EPIC BM25 performs comparably with docTTTTTquery with a 1.3× speedup.

Cost of pre-computing. We find that document vectors can be pre-computed for the MS-MARCO collection in approximately 14 hours on a single commodity GPU (GeForce GTX 1080ti). This is considerably less expensive than docTTTTTquery, which takes approximately 40 hours on a Google TPU (v3). When stored as half-precision (16-bit) floating point values, the vector for each document uses approximately 60KiB, regardless of the document length. This results in a total storage burden of approximately 500GiB for the entire collection. Pruning the collection to $r = 1000$ (which has minimal impact on ranking effectiveness) reduces the storage burden of each document to 3.9KiB (using 16-bit integer indices) and total storage to 34 GiB. The pruned document vectors use 34 GiB, when stored as 16-bit floating point values.

Interpretability of representations. A benefit of our approach is that the dimensions of the representation correspond to terms in the lexicon, allowing the representations to be easily inspected. In Figure 4.4, we present the relative scores for sample queries from MS-MARCO. We observe that the model is generally able to pick up on the terms that match intuitions of term importance. For instance,

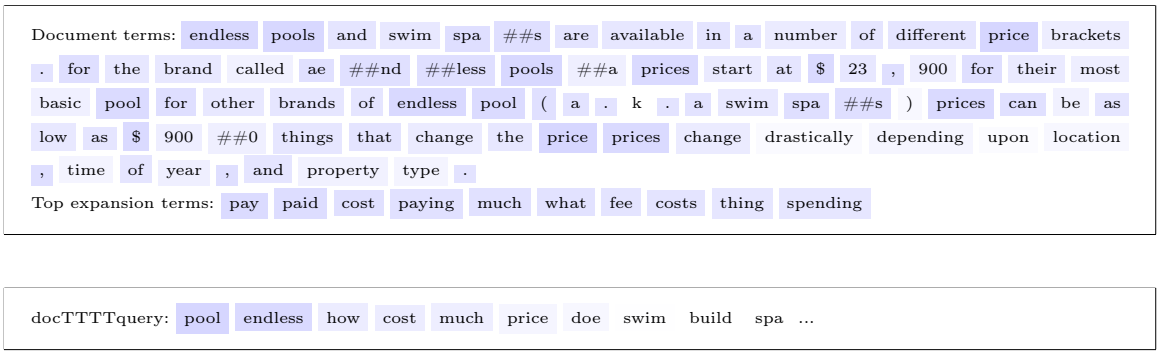

Figure 4.5 Relative EPIC representation values of terms that appear in a sample document. Alongside terms that appeared in the passage, the top 'expansion' terms are also shown. For reference, the most frequent terms produces by docTTTTTquery are also given, weighted by term frequency.

(a) gives highest scores to *california*, *aaa* (American Automobile Association), and *tow*. These three terms are good candidates for a keyword-based query with the same query intent. This approach does not necessarily just remove stop words; in (b) *what* is assigned a relatively high score. We provide an example of document vector importance scores in Figure 4.5. Because the document vector is dense, the figure only shows the terms that appear directly in the document and other top-scoring terms. Notice that terms related to *price*, *endless*, *pool(s)*, and *cost* are assigned the highest scores. In this case, the expansion of the term *cost* was critical for properly scoring this document, as a relevant query is *cost of endless pools/spas*. Although the terms that docTTTTTquery generate for this document are similar, the continuous values generated by EPIC paid off in a higher MRR@10 score for the query "*cost of endless pools/swim spa*" (a relevant question for this passage).

4.3.3 Summary

This section showed a design for an effective and inexpensive neural ranking architecture. This technique advances the art by further approaching fully BERT-based re-ranking performance, while providing low query-time latency and easy interpretability of representations. We also find that pruning can be an effective technique for reducing query latency without sacrificing effectiveness. This validates Hypothesis 2.2.

4.4 Discussion and Conclusions

In this chapter, I demonstrated that concerns surrounding the query-time efficiency of these models can be mitigated. Specifically, I showed that one can offload much of the computation to index-time in a way that substantially reduced the cost at query-time (Section 4.2). This validates Hypothesis 2.1. I then showed that this characteristic can be exploited to design a neural ranking architecture specifically designed with efficiency in mind: EPIC (Section 4.3). This model reduces the query-time cost to only query processing and a sparse dot product. Since the query processing can be done in parallel with initial retrieval, this approach can add as low as 6ms to query-time latency while still producing highly competitive ranking performance. This validates Hypothesis 2.2. Furthermore, the design of EPIC is interpretable, transparently assigning importance scores to query and document terms.

Chapter 5

Understanding Neural Ranking Behaviors

As shown in Chapter 3, pre-trained contextualized language models such as BERT [44] are effective at the task of ad-hoc retrieval. Despite this success, little is understood about *why* pretrained language models are effective for ad-hoc ranking. Previous work has shown that traditional IR axioms, e.g. that increased term frequency should correspond to higher relevance, do *not* explain the behavior of recent neural models [18]. Outside of IR, others have examined what characteristics contextualized language models learn in general [103, 106, 168], but it remains unclear if these qualities are valuable for ad-hoc ranking specifically. Thus, new approaches are necessary to characterize the models.

We propose a new framework aimed at Analyzing the Behavior of Neural IR ModeLs (ABNIRML), which aims to test the sensitivity of ranking models on specific textual properties. Probes consist of samples comprised of a query and two contrastive documents. We propose three strategies for building probes. The "measure and match" strategy (akin to the diagnostic datasets proposed by Rennings et al. [163]) constructs probing samples by controlling one measurement (e.g., term frequency) and varying another (e.g., document length) using samples from an existing IR collection. The "textual manipulation" strategy probes the effect that altering the text of a document text has on its ranking. The "dataset transfer" strategy constructs probes from non-IR datasets. The new probes allow us to isolate model characteristics—such as sensitivity

to word order, or preference for summarized rather than full documents—that cannot be analyzed using other approaches.

Using our new framework, we perform the first large-scale analysis of neural IR models. We compare today's leading ranking techniques, including those using BERT [44] and T5 [159], as well as methods focused on efficiency like DocT5Query [139] and EPIC (from Section 4.3). We present evidence that neural models use linguistic signals that are fundamentally different from classical term-matching metrics like BM25: when controlling for term frequency match, the neural models detect document relevance much more accurately than the BM25 baseline, and the effect is more pronounced in larger neural models. Further, unlike prior approaches, rankers based on BERT and T5 are heavily influenced by word order: shuffling the words in a document consistently lowers the document's score relative to the unmodified version, and neural rankers show a sensitivity to sentence order that is completely absent in classical models. We also find that the underlying pretrained language model alone does not determine a system's ad-hoc ranking behavior, and in particular the BERT-based EPIC model differs in many probes from a vanilla BERT re-ranker. Our battery of probes also uncover a variety of other findings, including that adding additional text to documents can often exhibit adverse behavior in neural models—decreasing the document's score when the added text is relevant, and increasing score when the added text is irrelevant.

In summary, we present a new framework (ABNIRML) for performing analysis of ad-hoc ranking models. We then demonstrate how the framework can provide insights into ranking model characteristics by providing the most comprehensive analysis of neural ranking models to date. Our released software framework facilitates conducting further analyses in future work (to be released upon acceptance).

5.1 Background and Preliminaries

Diagnostic datasets, proposed by Rennings et al. [163], reformulate traditional ranking axioms—e.g., that documents with a higher term frequency should receive a higher ranking score [50]—as empirical tests for analysing ranking models. Rennings et al. studied neural ranking architectures that predate the rise of contextualized language models for ranking, and focused on just four axioms. Câmara and Hauff [18] extended this work by adding five more previously-proposed ranking axioms (e.g., term proximity [194], and word semantics [49]) and evaluating on a distilled BERT model. They found that the axioms are inadequate to explain the ranking effectiveness of their model. Unlike these prior lines of work, we propose new probes that shed light onto possible sources of effectiveness, and test against current leading neural ranking architectures.

Although some insights about the effectiveness of contextualized language models for ranking have been gained using existing datasets [36] and indirectly through various model architectures [35, 70, 84, 110, 118, 138], they only provide circumstantial evidence. For instance, several works show how contextualized embedding similarity can be effective, but this does not imply that vanilla models utilize these signals for ranking. Rather than proposing new ranking models, in this work we analyze the effectiveness of existing models using controlled diagnostic probes, which allow us to gain insights into the particular behaviors and preferences of the ranking models.

Outside of the work in IR, others have developed techniques for investigating the behavior of contextualized language models in general. Although probing techniques [195] and attention analysis [185] can be beneficial for understanding model capabilities, these techniques cannot help us characterize and quantify the behaviors of neural ranking models. CheckList [164] and other challenge set techniques [124]

differ conceptually from our goals; we aim to characterize the behaviors to understand the qualities of ranking models, rather than provide additional measures of model quality.

5.2 Methodology

In order to characterize the behavior of ranking models we construct several diagnostic probes. Each probe aims to evaluate specific properties of ranking models and probe their behavior (e.g., if they are heavily influenced by term matching, discourse and coherence, conciseness/verbosity, writing styles, etc). We formulate three different approaches to construct probes (Measure and Match, Textual Manipulation, and Dataset Transfer).

In ad-hoc ranking, a query (expressed in natural language) is submitted by a user to a search engine, and a ranking function provides the user with a list of natural language documents sorted by relevance to the query. More formally, let $R(q, d) \in \mathbb{R}$ be a ranking function, which maps a given query q and document d (each being a natural-language sequence of terms) to a real-valued ranking score. At query time, documents in a collection D are scored using $R(\cdot)$ for a given query q, and ranked by the scores (conventionally, sorted descending by score). Learning-to-rank models optimize a set of parameters for the task of relevance ranking based on training data.

5.2.1 Document Pair Probing

We utilize a *document pair probing* strategy, in which probes are comprised of samples, each of which consists of a query and two documents that differ primarily in some characteristic of interest (e.g., textual elaboration). The ranking scores of the two documents are then compared (with respect to the query). This allows the isolation

of particular model preferences. For instance, a probe could consist of summarized and full texts of news articles; models that consistently rank full texts above summaries prefer elaborative text.

More formally, each document pair probe consists of a collection of samples S, where each $\langle q, d_1, d_2 \rangle \in S$ is a 3-tuple consisting of a query (or query-like text, q), and two documents (or document-like texts, d_1 and d_2). The relationship between d_1 and d_2 (with respect to q) for each sample defines the probe. For instance, a probe testing summarization could be defined as: (1) d_2 is a summary of d_1, and (2) d_1 is relevant to the query q.[1]

Each sample in the probe is scored as: $(+1)$ scoring d_1 above d_2 (a *positive* effect), (-1) scoring d_2 above d_1 (a *negative* effect), or (0) a *neutral* effect. Formally, the effect $\mathit{eff}(\cdot)$ of a given sample is defined as:

$$\mathit{eff}(q, d_1, d_2) = \begin{cases} 1 & R(q, d_1) - R(q, d_2) > \delta \\ -1 & R(q, d_1) - R(q, d_2) < -\delta \\ 0 & -\delta \leq R(q, d_1) - R(q, d_2) \leq \delta \end{cases} \tag{5.1}$$

The parameter δ adjusts how large the score difference between the scores of d_1 and d_2 must be in order to count as positive or negative effect. This allows us to disregard small changes to the score that are unlikely to affect the final ranking. In practice, δ depends on the ranking model because each model scores on different scales. Therefore we tune δ for each model (see Section 5.3.3)

[1]Note that the relationship between d_1 and d_2 must be asymmetric, or the probe is ill-defined. For example, paraphrasing is not a valid pair probe because both texts are paraphrases of one another, and it would therefore be ambiguous which to assign to d_1 and which to assign to d_2.

A model's performance on a particular probe is summarized by a single score s that averages the effect of all samples in the probe:

$$s = \frac{1}{|S|} \sum_{\langle q, d_1, d_2 \rangle \in S} \mathit{eff}(q, d_1, d_2) \tag{5.2}$$

Note that this score is in the interval $[-1, 1]$. Positive scores indicate a stronger preference towards documents from group 1 (d_1 documents), and negative scores indicate a preference towards documents from group 2 (d_2 documents). Scores near 0 indicate no strong preference or preferences that are split roughly evenly; disentangling these two cases requires analyzing individual effect scores.

There are several important differences between our setup and the "diagnostic dataset" approach proposed by Rennings et al. [163]. First, by including the δ threshold, we ensure that our probes measure differences that can affect the final order in ranked lists. Second, by including the "neutral effect" case in our scoring function, we distinguish between cases in which d_1 or d_2 are preferred and cases where neither document is strongly preferred. And finally, our probes are aimed at describing model behavior, rather than evaluating models.

5.2.2 Document Pair Probing Strategies

In this work, we explore three strategies for designing document pair probes. As discussed below, the strategies have different strengths and weaknesses. When used in concert, they allow our framework to characterize a wide variety of model behaviors. Figure 5.1 provides an overview of the strategies.

Measure and Match Probes (MMPs). Some surface-level characteristics of documents, such as its Term Frequency (TF) for a given query, are both easy to

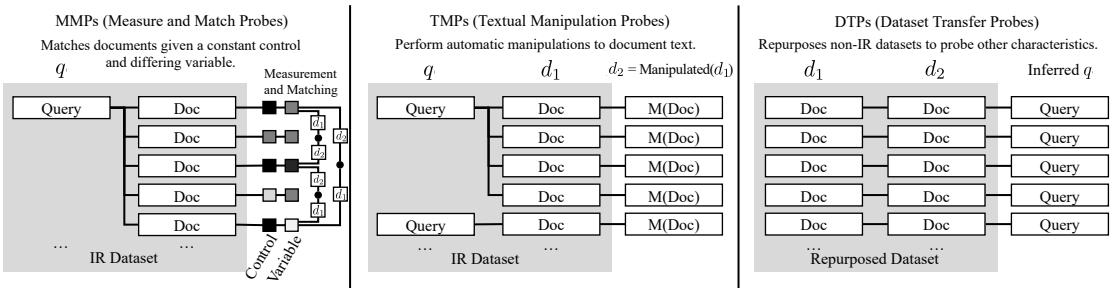

Figure 5.1 Overview of strategies for constructing probes. Each probe in ABNIRML is comprised of samples, each of which consists of a query (q) and two documents (d_1 and d_2).

measure and valuable for characterizing models.[2] By comparing the ranking scores of two documents in which such a characteristic differs (but are otherwise similar), one can gain empirical evidence into what factors influence a model. Measure and Match Probes (MMPs) follow such an approach. MMPs involve first measuring the characteristics of judged query-document pairs in an IR dataset. Then, the pairs are matched to form probe samples based on a *control* (a characteristic that matches between the documents, such as document length), and a *variable* (which differs between documents, such as TF). When matching control characteristics, a threshold can be employed to allow for more potential matches. MMPs have been explored by others [18, 163] by formulating existing ranking axioms[3] into empirical probes. Our MMP strategy generalizes the process for building such probes. Specifically, this encourages one to consider all combinations of controls and variables, promoting a more comprehensive analysis.

[2]In the case of TF, it has long been used as a core signal for ranking algorithms; a departure from monotonically increasing the score of a document as TF increases would represent a fundamental shift in the notion of relevance scoring [50].

[3]An example is TFC1 from [50], which suggests that higher TFs should be mapped to higher relevance scores.

Textual Manipulation Probes (TMPs). Not all characteristics are easily captured with MMPs. For instance, it would be difficult to probe the sensitivity to word order with MMPs; it is unlikely to find naturally-occurring documents that use the same words but in a different order, and even if identified, it would be unclear how to measure the quality of each word order. We overcome limitations like this by proposing Textual Manipulation Probes (TMPs). These probes apply a manipulation function to scored documents from an existing IR dataset. For probing word order, a simple manipulation function that shuffles the order of the words in the sentence can be used, which eliminates both the matching problem (all documents can be manipulated) and the measurement problem (the proposed manipulation function is destructive, almost certainly reducing the quality of the word order). Prior works that use a similar approach for probing ranking methods include the collection perturbation tests Fang et al. [51] (which perform operations like removing documents from the collection and deleting individual terms from documents) and a diagnostic dataset proposed by Rennings et al. [163] (which tests the effect of duplicating the document: an adaptation of a traditional ranking axiom). Although TMPs allow probing a wider variety of characteristics than MMPs, we note that they involve constructing artificial data; d_2 may not resemble documents seen in practice. Despite this, their versatility make TMPs an attractive choice for a variety of characteristics.

Dataset Transfer Probes (DTPs). Even with MMPs and TMPs, some characteristics may still be difficult to measure. For instance, if one wanted to probe the effect of text fluency (the degree to which language sounds like a native speaker wrote it) with a MMP, one would need an effective measure of text fluency and be able to match it to documents that otherwise provide similar text, which is a tall order. To probe fluency with a TMP, one would need a function that is able to consistently reduce (or improve) textual fluency, which is difficult to accomplish with today's techniques.

To probe characteristics like these, we propose Dataset Transfer Probes (DTPs). In this setting, a dataset built for a purpose other than ranking is repurposed to probe a ranking model's behavior. By using a DTP, one could use a dataset of human-written textual fluency pairs (e.g., from the JFLEG dataset [130]) to sidestep challenges in both measurement and manipulation. Text pair datasets are abundant, allowing us to probe a wide variety of characteristics, like fluency, formality, and summarization. With these probes, d_1 and d_2 can be easily defined by the source dataset. In some cases, external information can be used to infer a corresponding q, such as using the title of the article as a query for news article summarization tasks, a technique that has been studied before to train ranking models [112]. In other cases, queries can be artificially generated.

5.3 Experiment

5.3.1 Datasets

We use the MS-MARCO passage dataset [19] to train the neural ranking models. The training subset contains approximately $809k$ natural-language questions from a query log (with an average length of 7.5 terms) and 8.8 million candidate answer passages (with an average length of 73.1 terms). Due to its scale in number of queries, it is shallowly annotated, almost always containing fewer than 3 positive judgments per query. This dataset is frequently used for training neural ranking models, and has been shown to effectively transfer relevance signals to other collections [136].

We build MMPs and TMPs using the TREC Deep Learning 2019 passage dataset [34]. Although it uses the MS-MARCO passage collection, TREC DL has fewer queries (containing only 43 queries with relevance judgments). However, it has much deeper relevance judgments (on average, 215 per query). The judgments are

also graded as highly relevant (7%), relevant (19%), topical (17%), and non-relevant (56%), allowing us to make more fine-grained comparisons. We opt to perform our analysis in a passage ranking setting to eliminate effects of long document aggregation, an area with many model varieties that is still under active investigation [95].

5.3.2 Models

We compare a sample of several models covering a traditional lexical model (BM25), a conventional learning-to-rank approach (LightGBM), and neural models with and without contextualized language modeling components. We also include two models that focus on query-time computational efficiency. The neural models represent a sample of the recent state-of-the-art ranking models.

BM25. We use the Terrier [143] implementation of BM25 with default parameters. BM25 is an unsupervised model that incorporates lexical the features of term frequency (TF), inverse document frequency (IDF), and document length. (TREC DL 2019 MRR: 0.7851.)

L-GBM [83]. We use the Light Gradient Boosting Machine model currently used by the Semantic Scholar search engine [52].[4] This public model was trained on click-through data from this search engine, meaning that it services various information needs (e.g., navigational and topical queries). A wide variety of features are used, not all of which are available in our ranking setting (e.g., recency, in-links, etc.). Thus, we only supply the text-based features L-GBM uses like lexical overlap and scores from a light-weight language model [63]. This serves as a non-neural learning-to-rank baseline. (TREC DL 2019 MRR: 0.7589.)

KNRM and **C-KNRM** [37, 214] are kernel-based ranking models that calculate the cosine similarity between the word embeddings of query and document terms,

[4] `https://github.com/allenai/s2search`

aggregating the scores using Gaussian kernels. The Convolutional variant (C-KNRM) applies 1-dimensional convolutions to build n-gram representations. For both models, we use default settings (e.g., 11 buckets, 3-grams), and train the models using the official training sequence of the MS-MARCO passage ranking dataset. (KNRM TREC DL 2019 MRR: 0.6876; C-KNRM TREC DL 2019 MRR: 0.7392.)

BERT [44]. We use the Vanilla BERT model, introduced in Section 3.2. We fine-tune the `bert-base-uncased` model for this task using the official training sequence of the MS-MARCO passage ranking dataset. (TREC DL 2019 MRR: 0.9368.)

T5 [159]. The Text To Text Transformer ranking model [136] scores documents by predicting whether the concatenated query, document, and control tokens is likely to generate the term 'true' or 'false' as indication of relevance. We use the model weights released by the authors, which were tuned on the MS-MARCO passage ranking dataset. At the time of writing, T5 tops several ad-hoc ranking leaderboards. (TREC DL 2019 MRR: 0.9671.)

EPIC, introduced in Section 4.3 We use the `bert-base-uncased` model, and tune the model for ranking using the train split of the MS-MARCO passage ranking dataset. (TREC DL 2019 MRR: 0.8891.)

DT5Q [136]. The T5 variant of the Doc2Query model (DT5Q) generates additional terms to add to documents based using a T5 model. The expanded document can be efficiently indexed, boosting the weight of terms likely to match queries. We use the model released by the authors, which was trained using the MS-MARCO passage training dataset. For our probes, we generate four queries to add to each document. As was done in the original paper, we use BM25 a a scoring function over the expanded documents. (TREC DL 2019 MRR: 0.8631.)

Table 5.1 Results of Measure and Match Probes (MMPs) on TREC DL 2019.

Variable	Control	BM25	L-GBM	DT5Q	KNRM	C-KNRM	EPIC	BERT	T5	Samples
Relevance	Length	+0.40	+0.40	+0.47	+0.34	+0.39	+0.55	+0.58	+0.58	19,676
	TF	−0.03	+0.04	+0.08	+0.05	+0.13	+0.29	+0.34	+0.41	31,619
	Overlap	+0.41	+0.34	+0.45	+0.29	+0.35	+0.51	+0.55	+0.57	4,762
Length	Relevance	−0.05	+0.04	−0.08	−0.04	−0.02	+0.05	* −0.01	+0.06	515,401
	TF	−0.14	+0.02	−0.08	−0.10	* −0.02	+0.03	−0.09	+0.13	88,582
	Overlap	+0.51	+0.26	+0.22	+0.14	+0.15	+0.29	+0.20	+0.21	3,963
TF	Relevance	+0.88	+0.50	+0.74	+0.42	+0.43	+0.49	+0.41	+0.40	303,058
	Length	+1.00	+0.59	+0.84	+0.55	+0.56	+0.58	+0.54	+0.52	19,770
	Overlap	+0.80	+0.37	+0.32	+0.22	+0.23	+0.39	+0.26	+0.26	2,294
Overlap	Relevance	+0.70	+0.20	+0.53	+0.22	+0.22	+0.16	+0.19	+0.19	357,470
	Length	+0.75	+0.35	+0.59	+0.36	+0.36	+0.31	+0.31	+0.32	20,819
	TF	+0.88	−0.03	+0.48	* +0.06	* +0.02	* −0.01	+0.11	* +0.02	13,980

Positive scores indicate a preference towards a higher value of the variable. Score marked with * are not statistically significant (see Section 5.3.4).

5.3.3 Choosing Delta

Recall that δ indicates the minimum absolute difference of scores in a document pair probe to have a positive or negative effect. Since each model scores documents on a different scale, we empirically choose a δ per model. We do this by first scoring the top 100 documents retrieved by BM25 for the MS-MARCO dev collection. Among the top 10 results, we calculate the differences between each adjacent pair of scores (i.e., $\{R(q,d_1) - R(q,d_2), R(q,d_2) - R(q,d_3), ..., R(q,d_9) - R(q,d_{10})\}$, where d_i is the ith highest scored document for q). We set δ to the median difference among all queries in the dev collection. By setting the threshold this way, we can expect the differences captured by the probes to have an effect on the final ranking score *at least* half the time. In practice, we see relatively few differences when using alternative approaches for choosing δ (e.g., using the mean difference).

5.3.4 Significance Testing

We use a two-sided paired T-Test to determine the significance (pairs of $R(q, d_1)$ and $R(q, d_2)$). We use a Bonferroni correction over each table to correct for multiple tests, and test for $p < 0.001$.

5.3.5 Software and Libraries

We use the following software libraries and packages to conduct our analysis: Anserini [217], PyTerrier [120], OpenNIR [107], and HuggingFace Transformers [211].

5.4 Analysis

5.4.1 Measure and Match Probes (MMPs)

Recall that MMPs measure a characteristic about a document and match them with documents that have a differing characteristic given a control. We explore the following characteristics:

- **Relevance**: the human-assessed graded relevance score of a document to the given query.
- **Length**: the document length, in total number of non-stopword tokens.
- **TF**: the individual Porter-stemmed Term Frequencies of non-stopword terms from the query. To determine when the TF of two documents are different, we use the condition that the TF of at least one query term in d_1 must be greater than the same term in d_2, and that no term in d_1 can have a lower TF than the corresponding term in d_2.
- **Overlap**: the proportion of non-stopword terms in the document that appear in the query. Put another way, the total TF divided by the document length.

Each of these characteristics can be used as both a variable (matching based on differing values) and a control (matching based on identical values). We examine all pairs of these characteristics, greatly expanding upon IR axioms investigated in prior work. The results using the TREC DL 2019 dataset are available in Table 5.1. Positive scores indicate a preference for a higher variable, and negative scores indicate a preference for a lower variable. We now highlight key findings.

EPIC, BERT, and T5 can distinguish relevance grades when TF is held constant. It appears that the model's capacity is related to its effectiveness in this setting: T5 (+0.41) performs better than BERT (+0.34), which performs better than EPIC (+0.29). Past models are not able to do make make these relevance distinctions nearly as effectively (at best, +0.13 for C-KNRM). EPIC, BERT, and T5 also perform better at distinguishing relevance grades than the other models when length and overlap are held constant, though by a lesser margin.

Models generally have similar sensitivity to document length, TF, and overlap. With the exception of models that use BM25 for scoring (BM25 and DT5Q), all the models we explore have similar behaviors when varying length, TF, and overlap. This suggests that although signals like TF are not *required* for EPIC, BERT, and T5 to rank effectively, they still remain an important signal when available.

5.4.2 Textual Manipulation Probes (TMPs)

As we saw in Section 5.4.1, the ranking models that use contextualized language models are able to effectively distinguish relevance when controlled for various traditional relevance signals. We now use TMPs to investigate alternative explanations for the performance. Recall that TMPs perform automatic manipulations to document text. We perform a variety of probes, presented in Table 5.2, with an overall score

Table 5.2 Results of Text Manipulation Probes (TMPs) on TREC DL 2019.

	BM25	L-GBM	DT5Q	KNRM	C-KNRM	EPIC	BERT	T5	Samples
Remove Stops/Punct.	* 0.00	−0.20	−0.08	−0.43	−0.60	−0.94	+0.18	−0.83	9,259
$rel \in \{0,1\}$	* 0.00	−0.19	−0.07	−0.44	−0.61	−0.93	+0.34	−0.83	6,758
$rel \in \{2,3\}$	* 0.00	−0.22	−0.10	−0.41	−0.58	−0.96	−0.26	−0.81	2,501
Lemmatize	* 0.00	−0.02	* +0.02	* +0.01	−0.10	−0.10	−0.04	−0.59	9,259
$rel \in \{0,1\}$	* 0.00	−0.02	* +0.02	* 0.00	−0.11	−0.10	* −0.01	−0.59	6,758
$rel \in \{2,3\}$	* +0.01	−0.02	* +0.02	* +0.05	−0.06	−0.11	−0.14	−0.58	2,501
Shuf. Words	* 0.00	−0.25	−0.10	* 0.00	−0.11	−0.82	−0.38	−0.72	9,260
$rel \in \{0,1\}$	* 0.00	−0.21	−0.06	* 0.00	−0.07	−0.78	−0.28	−0.67	6,759
$rel \in \{2,3\}$	* 0.00	−0.36	−0.20	* 0.00	−0.23	−0.90	−0.66	−0.85	2,501
Shuf. Prepositions	+0.01	−0.02	* +0.03	−0.01	+0.03	−0.07	−0.01	−0.61	9,239
$rel \in \{0,1\}$	* 0.00	−0.01	* +0.04	−0.01	* +0.04	−0.06	* +0.01	−0.61	6,743
$rel \in \{2,3\}$	+0.05	−0.02	* 0.00	* −0.02	* +0.02	−0.08	−0.06	−0.60	2,496
Shuf. Sent. Order	* 0.00	* 0.00	−0.04	* 0.00	+0.02	−0.18	−0.13	−0.16	7,295
$rel \in \{0,1\}$	* 0.00	* 0.00	* −0.04	* 0.00	* +0.03	−0.16	−0.10	−0.12	5,249
$rel \in \{2,3\}$	* 0.00	* 0.00	* −0.03	* 0.00	* +0.01	−0.24	−0.18	* −0.18	2,058
Replace with Query	+0.99	+1.00	+1.00	+1.00	+1.00	+0.42	+0.98	+0.67	9,260
$rel \in \{0,1\}$	+0.99	+1.00	+1.00	+1.00	+1.00	+0.55	+0.99	+0.82	6,759
$rel \in \{2,3\}$	+0.99	+1.00	+1.00	+1.00	+1.00	+0.08	+0.96	+0.27	2,501
Add DocT5Query Terms	+0.33	+0.41	+0.21	+0.37	−0.02	* −0.19	−0.22	−0.47	9,260
$rel \in \{0,1\}$	+0.32	+0.37	+0.20	+0.33	−0.05	−0.20	−0.26	−0.42	6,759
$rel \in \{2,3\}$	+0.34	+0.53	+0.22	+0.47	+0.09	* −0.15	* −0.11	−0.63	2,501
Add non-rel sent.	−0.03	+0.20	+0.01	−0.02	+0.05	+0.46	+0.24	+0.43	9,260
$rel \in \{0,1\}$	+0.06	+0.23	+0.09	+0.04	+0.12	+0.52	+0.29	+0.51	6,759
$rel \in \{2,3\}$	−0.18	+0.15	* −0.11	−0.12	−0.08	+0.34	+0.13	+0.27	2,501

Positive scores indicate a preference for the manipulated document text; negative scores prefer the original text. Score marked with * are not statistically significant (see Section 5.3.4). $rel \in \{0,1\}$ are at best "topical", and $rel \in \{2,3\}$ are relevant or highly relevant.

and scores broken down by non-relevant ($rel \in \{0,1\}$) and relevant ($rel = \{2,3\}$) documents. We now highlight our key findings.

EPIC, BERT, and T5 tend to be adversely affected by traditional pre-processing steps. When removing stop words and punctuation from documents, we find that this negatively impacts most models, with the largest effect seen on EPIC and T5. The BERT model actually prefers this modification for non-relevant documents, though it reduces the score of relevant documents. This behavior is also particularly interesting because it is not exhibited by EPIC, even though it uses the

same contextualized model base. We note that in cases where there is a disparity like this (i.e., scores for relevant documents and non-relevant documents differ), the model is applying different behavior at different relevance grades. In this case, the behavior suggests that the presence of stop words and punctuation are, at least in part, used as a signal that distinguishes relevant grades by BERT. To probe the effect of stemming, we use SpaCy's [71] lemmatizer to remove word inflections, rather than a stemmer, because the results from a stemming function like Porter are often not found in the lexicon of models like BERT. T5 is most affected by this change (-0.59). However, BERT exhibits different preferences for relevant and non-relevant documents: there is no significant effect for non-relevant documents, and for relevant documents, stemming yields a score of -0.14.

EPIC, BERT, and T5 are strongly affected by word and sentence order. Many ranking models make a bag-of-words assumption; that is, that the order of the terms in the query and document do not matter. We probe the effect of word order by randomly shuffling the words in each document as our next TMP. As seen in Table 5.2, shuffling the words in a document can have a substantial effect on the ranking scores for EPIC, BERT, and T5. We see here again that the impact is larger for higher relevance levels, suggesting that word order is an important signal for distinguishing relevance. Note that the magnitude of these scores is higher than those for MMPs, signaling that word order can have a larger impact on model scores than traditional signals like TF.

To control for *local* word order (i.e., term adjacency), which is captured by n-gram approaches, we conduct another TMP that shuffles the order of the sentences in a document. (Documents containing only a single sentence are omitted from this probe.) Here, we see that an effect remains for EPIC, BERT, and T5, though it is substantially reduced (and not significant, in some cases for T5). This suggests that

discourse-level signals (e.g., what topics are discussed earlier in a document) have some effect on the models, or the models encode some positional bias (e.g., preferring answers at the start of documents). We also find that shuffling the prepositions in a text has a large effect on T5, but not for other models, suggesting that the T5 models uses the relation of terms to one another in the text as an important signal. Overall, the shuffling results indicate that ranking models like EPIC, BERT, and T5 make use of word order, and that the importance or word order is correlated with the degree of relevance.

All models behave navigationally. In some applications, it is desirable to rank exact query matches highly (e.g., navigational queries), but in other applications users would expect further elaboration on the topic they are searching for (e.g., question answering). To probe this behavior, our next TMP uses a manipulation function that replaces the document with the query text itself. We observe that nearly all systems are heavily biased towards these exact query matches. The exceptions are EPIC and T5, which favor the query text less frequently (especially for relevant documents). We note that although EPIC is close to a score of 0, this is mostly due to the positive effects (1,321 of 2,501) and negative effects (1,129 of 2,501) cancelling one another out, rather than neutral effects (i.e., falling within the δ range).

EPIC, BERT, and T5 behave unexpectedly when additional content is introduced in documents. We conduct two TMPs that involve adding text to documents: one that appends expansion terms from the DocT5Query model to the document, one that appends non-relevant text to the document (by sampling a sentence from a $rel = 0$ document). Models that rely heavily on unigram matching (e.g., BM25) respond positively to the addition of DocT5Query terms. Even the DocT5Query model itself sees an additional boost, suggesting that weighting the expansion terms higher in the document may further improve the effectiveness of this

Table 5.3 Results of Dataset Transfer Probes (DTPs).

Probe	Dataset	BM25	L-GBM	DT5Q	KNRM	C-KNRM	EPIC	BERT	T5	Samples
Fluency	JFLEG	+0.03	* 0.00	* +0.04	* −0.01	−0.04	+0.19	+0.10	+0.12	5,069
	spellchecked	* +0.01	* 0.00	* −0.01	* −0.01	* +0.02	+0.16	+0.07	+0.21	5,183
Formality	GYAFC	−0.02	−0.09	−0.07	* +0.03	+0.13	+0.10	−0.05	+0.10	6,850
	entertainment	+0.04	−0.04	* 0.00	* +0.05	+0.17	+0.22	* +0.05	+0.19	3,149
	family	−0.08	−0.13	−0.13	* 0.00	+0.09	* 0.00	−0.13	* +0.03	3,701
Summ.	XSum	+0.66	+0.19	+0.66	+0.42	+0.45	+0.38	+0.49	−0.03	17,959
	CNN	+0.37	−0.43	+0.40	−0.22	−0.05	+0.45	+0.16	−0.64	7,254
	Daily Mail	* −0.02	−0.79	+0.06	−0.59	−0.45	+0.52	−0.15	−0.68	19,081

Positive scores indicate a preference for fluent, formal, and summarized text. Score marked with * are not statistically significant (see Section 5.3.4).

model. However, EPIC, BERT, and T5 often respond *negatively* to these additions. We also find that adding non-relevant sentences to the end of relevant documents often increases the ranking score of EPIC, BERT, and T5. This is in contrast with models like BM25, in which the scores of relevant documents decrease with the addition of non-relevant information. From the variable length MMPs, we know that this increase in score is likely not due to increasing the length alone. Such characteristics may pose a risk to ranking systems based on EPIC, BERT, or T5, in which content sources could aim to increase their ranking simply by adding non-relevant content to the end of their documents.

5.4.3 Dataset Transfer Probes (DTPs)

We now explore model behaviors using DTPs. Recall that a DTP repurposes a non-IR dataset as a diagnostic tool by using it to construct probes. In this work, we explore three characteristics.

EPIC, BERT, and T5 slightly favor more fluent text. From the TMPs, we found that models are highly influenced by word order and inflectional endings. One possible explanation is that these models incorporate signals of textual fluency. To test this hypothesis directly, we propose a DTP using the JFLEG dataset [130]. This dataset contains sentences from English-language fluency tests. Each non-fluent sentence is corrected for fluency by four fluent English speakers to make the text sound 'natural' (changes include grammar and word usage changes). We treat each fluent text as a d_1 paired with the non-fluent d_2. We generate q by randomly selecting a noun chunk that appears in both versions of the text. (If no such chunk exists, we discard the sample.) The results for this DTP are shown in Table 5.3. We observe that the models most substantially impacted by shuffling words in a text (EPIC, BERT, and T5) are also most affected here. However, the magnitude of the effect is much lower, considering that the corrections are often rather minor. We see similar results when controlling for spelling (i.e., when both versions use correct spelling). We note that in this case, a large proportion of the effects are neutral (i.e., within δ, meaning that they may not affect the ordering of ranked lists). The fluency experiments indicate that the shuffling results from the TMPs may be partially explained by the fact that they reduce the fluency of the text, and that fluency can have an effect on model scores.

EIPC and BERT slightly favor formal text. The style of a text may also have an influence on learning-to-rank models, perhaps due to bias in training data labeling (an annotator may be more likely to select an answer that is written more formally) or the pre-training objective (e.g., BERT is trained on books and Wikipedia text, both of which are more formal than much other text online). We probe this by building a DTP from the GYAFC dataset [161]. This dataset selects sentences from Yahoo Answers and has four annotators make edits to the text that either improve the

formality (for text that is informal), or reduce the formality (for text that is already formal). We treat formal text as d_1 and informal text as d_2. Since the text came from Yahoo Answers, we can link the text back to the original questions using the Yahoo L6 dataset[5]. We treat the question (title) as q. In cases where we cannot find the original text or there are no overlapping non-stopword lemmas from q in both d_1 and d_2, we discard the sample. Results from this DTP are shown in Table 5.3, split by forum. Positive scores indicate a preference for formal text. We observe that EPIC and T5 exhibit similar behaviors (preferring formal text from the entertainment section, and having no style preference in the family section). This is in contrast with BERT's behavior, which has no significant impact on entertainment, but a preference towards *informal* text for family. These probes show that these ranking methods can be affected by the formality of the document, a quality usually not exhibited by prior models.

Model behaviors vary considerably for summarization. Recall from the MMPs in Section 5.4.1 that most models do not have strong biases for document lengths. This raises the question: to what extent does the *verbosity* of the text matter to ranking models? To probe this, we construct DTPs from summarization datasets. Intuitively, a text's summary will be less verbose than its full text. We utilize two datasets to conduct this probe: XSum [131], and CNN/DailyMail [183]. The former uses extremely concise summaries from BBC articles, usually consisting of a single sentence. The CNN/DailyMail dataset uses slightly longer bullet point list summaries, usually consisting of around 3 sentences. For these probes, we use the title of the article as q, the summarized text as d_1, and the article body as d_2. When there is no overlap between the non-stopword lemmas of q in both d_1 and d_2, we discard the samples. We further sub-sample the dataset at 10% because the datasets are already

[5] `https://webscope.sandbox.yahoo.com/catalog.php?datatype=l&did=11`

rather large. To handle the long full text in BERT and EPIC, we use the passage aggregation strategy proposed by [110]. Results from the summarization DTPs are shown in Table 5.3. Positive scores indicate a preference for the summarized text. Here, we observe some interesting behaviors. First, BM25 has a strong (+0.66) preference for the summaries in XSum, a moderate preference for summaries in CNN (+0.37), and no significant preference for Daily Mail. This suggests different standards among the various datasets, e.g., XSum (BBC) must use many of the same terms from the titles in the summaries, and provide long documents (reducing the score) that may not repeat terms from the title much. The preference for summaries in XSum can be seen across all models except T5, which very slightly favors the full text. The behaviors for the CNN and Daily Mail DTPs vary considerably across models. For instance, EPIC prefers summaries for both (+0.45 and +0.52, respectively), and T5 prefers full text for both (-0.64 and -0.68). These discrepancies warrant exploration in future work.

Correcting model biases may not be easy. To determine whether models can overcome the biases we observe in our diagnostics, we conduct a test where we augment the training process of a model by introducing additional training pairs. We use the EPIC model and augment the training process with Daily Mail pairs (favoring full text) from the 90% of articles not used in our evaluation. After testing several ratios between MS-MARCO relevance pairs and Daily Mail pairs, we found a 3:1 ratio to be a good balance between maintaining similar ranking performance (MRR of 0.8795 on TREC DL 2019, down from 0.8891) and reducing the bias toward summaries (+0.35 on Daily Mail down from +0.52). The training procedure also reduced summarization bias on XSum (+0.16) and CNN (+0.40). This experiment demonstrates that correcting biases identified by ABNIRML may not be easy, and deserves future work.

5.5 Discussion and Conclusions

We presented a new framework (ABNIRML) for analyzing ranking models based on three probing strategies. By using probes from each strategy, we demonstrated that a variety of insights can be gained about the behaviors of recently-proposed ranking models, such as those based on BERT and T5. Our analysis is, to date, the most extensive analysis of the behaviors of neural ranking models, and sheds light on several unexpected model behaviors. For instance, adding non-relevant text can increase a document's ranking score, even though the models are largely not biased towards longer documents. We also see that the same base language model used with a different ranking architecture can yield different behaviors, such as higher sensitivity to shuffling a document's text. Meanwhile, different language models can be sensitive to different characteristics, such as the importance of prepositions. These results validate Hypothesis 3. We also observed that correcting these biases may be non-trivial.

Chapter 6

Conclusions

In this dissertation, I presented evidence that neural networks can be trained to perform ad-hoc search effectively, and they can do so with reasonable query-time latency. This evidence is backed up by following scholarly publications [108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119]. I first demonstrated that neural techniques can produce effective ranking models (Hypothesis 1). Neural models can be trained both in cross-task (weak supervision) and cross-domain (web passage to scientific literature) transfer settings (Hypothesis 1.1). Then I showed that dataset characteristics can be easily incorporated into neural ranking as both features and in the training process (Hypothesis 1.2). I also demonstrated that using contextualized language modeling resources, such as BERT, can be beneficial for ranking, both when used directly as a ranker and incorporated into existing models (Hypothesis 1.3). I then validated Hypothesis 1.4 by showing that relevance signals can be effectively transferred across languages, benefiting search for languages with fewer resources. To address query-time latency concerns (Hypothesis 2), I showed that much of the computation for contextualized language models can be offloaded to index-time (Hypothesis 2.1). I also proposed and demonstrated a ranking model's effectiveness that was specifically designed to reduce the query-time computational cost (Hypothesis 2.2). I wrapped up by presenting a new approach for probing the linguistic biases exhibited by these new models. I found that although they provide superior ranking effectiveness, they show some unexpected side-effects (Hypothesis 3).

The works covered in this dissertation came during a paradigm shift in ad-hoc search. Specifically, the advent of contextualized language models has enabled much more effective models while also introducing new challenges. Work in this dissertation provided foundational explorations in a variety of directions. To name a few, CEDR (Section 3.2) was among the first works to demonstrate the effectiveness of contextualized language models for *document* (rather than *passage*) ranking, and it was the first to use a representation-based passage aggregation strategy. Passage aggregation continues to be an active area of research, with representation-based approaches being the most effective [95]. The cross-lingual ranking experiments demonstrated in Section 3.4 were the first to show how neural ranking models trained on only English relevance data can greatly benefit search in other languages, and ABNIRML (Chapter 5) was the first work to identify several linguistic biases exhibited neural ranking models. Finally, the works in Chapter 4 were among the first to explore computational efficiency concerns of using these models for ranking. They provided both an effective framework for reducing the cost of any transformer-based model and a concrete model architecture that was able to exploit this framework.

Finally, the contributions of this dissertation extend also to several open-source software tools. OpenNIR[1] [107] provided an end-to-end system for performing neural re-ranking experiments. And `ir_datasets`[2] provides a fast and robust interface to over a dozen standard IR datasets and benchmarks. Others in the community are adopting these tools to run experiments and access data.

[1] `https://opennir.net/`

[2] `https://ir-datasets.com/`

Bibliography

[1] Google year in search. https://archive.google.com/trends/2014/, 2014. Accessed: 2020-01-02.

[2] Google's search algorithm and ranking system. https://www.google.com/search/howsearchworks/algorithms/, 2020. Accessed: 2020-01-02.

[3] Zeynep Akkalyoncu Yilmaz, Wei Yang, Haotian Zhang, and Jimmy Lin. Cross-domain modeling of sentence-level evidence for document retrieval. In *EMNLP*, 2019.

[4] Gianni Amati and Cornelis Joost Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *TOIS*, 2002.

[5] Arash Ardakani, Zhengyun Ji, Sean C Smithson, Brett H Meyer, and Warren J Gross. Learning recurrent binary/ternary weights. In *ICLR*, 2019.

[6] Nima Asadi, Donald Metzler, Tamer Elsayed, and Jimmy Lin. Pseudo test collections for learning web search ranking functions. In *SIGIR*, 2011.

[7] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A nucleus for a web of open data. In *The Semantic Web*, 2007.

[8] Leif Azzopardi, Maarten de Rijke, and Krisztian Balog. Building simulated queries for known-item topics: An analysis using six european languages. In *SIGIR*, 2007.

[9] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. Ms marco: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268, 2016.

[10] Iz Beltagy, Arman Cohan, and Kyle Lo. Scibert: Pretrained contextualized embeddings for scientific text. *CoRR*, abs/1903.10676, 2019.

[11] Yoshua Bengio, Jerome Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, 2009.

[12] Richard Berendsen, Manos Tsagkias, Wouter Weerkamp, and Maarten de Rijke. Pseudo test collections for training and tuning microblog rankers. In *SIGIR*, 2013.

[13] Kurt D. Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 2008.

[14] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013.

[15] Martin Braschler. 2003–overview of results. In *Workshop of the Cross-Language Evaluation Forum for European Languages*, 2003.

[16] Martin Braschler, Peter Schäuble, and Carol Peters. Cross-language information retrieval (clir) track overview. In *TREC*, 2000.

[17] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581), 2010.

[18] Arthur Câmara and Claudia Hauff. Diagnosing bert with retrieval heuristics. In *ECIR*, 2020.

[19] Daniel Fernando Campos, T. Nguyen, M. Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, L. Deng, and Bhaskar Mitra. Ms marco: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268, 2016.

[20] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *ICML*, 2007.

[21] Claudio Carpineto and Giovanni Romano. A survey of automatic query expansion in information retrieval. *Acm Computing Surveys (CSUR)*, 2012.

[22] Xinlei Chen and Abhinav Gupta. Weakly Supervised Learning of Convolutional Networks. In *International Conference on Computer Vision*, 2015.

[23] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What Does BERT Look At? An Analysis of BERT's Attention. In *BlackBoxNLP @ ACL*, 2019.

[24] Charles L. A. Clarke, Nick Craswell, and Ian Soboroff. Overview of the TREC 2009 web track. In *TREC*, 2009.

[25] Arman Cohan, Luca Soldaini, Nazli Goharian, Allan Fong, Ross Filice, and Raj Ratwani. Identifying significance of discrepancies in radiology reports. In *SIAM*

International Conference on Data Mining (SDM) - Workshop on Data Mining for Medicine and Healthcare (DMMH), 2016.

[26] Thomas F. Coleman and Zhijun Wu. Parallel Continuation-based Global Optimization for Molecular Conformation and Protein Folding. *Journal of Global Optimization*, 8(1):49–65, January 1996.

[27] Kevyn Collins-Thompson, Craig Macdonald, Paul Bennett, Fernando Diaz, and Ellen M Voorhees. TREC 2014 web track overview. Technical report, MICHIGAN UNIV ANN ARBOR, 2015.

[28] Ronan Collobert, J. Weston, L. Bottou, Michael Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Machine Learning Research*, 12, 2011.

[29] Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. *CoRR*, abs/1710.04087, 2017.

[30] Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *SIGIR*, 2009.

[31] Gordon V. Cormack, Mark D. Smucker, and Charles L. A. Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *Information Retrieval*, 14, 2010.

[32] Nick Craswell, W Bruce Croft, Jiafeng Guo, Bhaskar Mitra, and Maarten de Rijke. Neu-IR: The SIGIR 2016 workshop on neural information retrieval. In *SIGIR*, 2016.

[33] Nick Craswell, W Bruce Croft, Maarten de Rijke, Jiafeng Guo, and Bhaskar Mitra. SIGIR 2017 workshop on neural information retrieval (Neu-IR'17). In *SIGIR*, 2017.

[34] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. Overview of the TREC 2019 deep learning track. In *TREC*, 2019.

[35] Zhuyun Dai and Jamie Callan. Context-aware sentence/passage term importance estimation for first stage retrieval. *CoRR*, abs/1910.10687, 2019.

[36] Zhuyun Dai and Jamie Callan. Deeper text understanding for ir with contextual neural language modeling. In *SIGIR*, 2019.

[37] Zhuyun Dai, Chenyan Xiong, James P. Callan, and Zhiyuan Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *WSDM*, 2018.

[38] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *ACL*, 2019.

[39] Joachim Daiber, Max Jakob, Chris Hokamp, and Pablo N. Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th International Conference on Semantic Systems*, 2013.

[40] Jeffrey Dalton, Laura Dietz, and James Allan. Entity query feature expansion using knowledge base links. In *SIGIR*, 2014.

[41] Domenico Dato, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. Fast ranking with additive ensembles of oblivious and non-oblivious regression trees. *ACM Transactions on Information Systems*, 35(2), 2016.

[42] Mostafa Dehghani, Arash Mehrjou, Stephan Gouws, Jaap Kamps, and Bernhard Schölkopf. Fidelity-weighted learning. In *ICLR*, 2017.

[43] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W Bruce Croft. Neural ranking models with weak supervision. In *SIGIR*, 2017.

[44] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

[45] Laura Dietz and Ben Gamari. TREC CAR: A data set for complex answer retrieval (version 1.5), 2017. URL `http://trec-car.cs.unh.edu`.

[46] Laura Dietz, Manisha Verma, Filip Radlinski, and Nick Craswell. TREC complex answer retrieval overview. In *TREC*, 2017.

[47] Laura Dietz, Manisha Verma, Filip Radlinski, and Nick Craswell. TREC complex answer retrieval overview. In *Proceedings of TREC*, 2017.

[48] Benjamin Van Durme, Pushpendre Rastogi, Adam Poliak, and M. Patrick Martin. Efficient, compositional, order-sensitive n-gram embeddings. In *EACL*, 2017.

[49] Hui Fang and ChengXiang Zhai. Semantic term matching in axiomatic approaches to information retrieval. In *SIGIR '06*, 2006.

[50] Hui Fang, T. Tao, and ChengXiang Zhai. A formal study of information retrieval heuristics. In *SIGIR*, 2004.

[51] Hui Fang, T. Tao, and ChengXiang Zhai. Diagnostic evaluation of information retrieval models. *ACM Trans. Inf. Syst.*, 29, 2011.

[52] Sergey Feldman. Building a better search engine for semantic scholar, Jul 2020. URL https://medium.com/ai2-blog/building-a-better-search-engine-for-semantic-scholar-ea23a0b661e7.

[53] Nicola Ferro, Claudio Lucchese, Maria Maistro, and Raffaele Perego. Continuation Methods and Curriculum Learning for Learning to Rank. In *CIKM*, 2018.

[54] Zhe Gan, Yunchen Pu, Ricardo Henao, Chunyuan Li, Xiaodong He, and Lawrence Carin. Learning generic sentence representations using convolutional neural networks. In *EMNLP*, 2016.

[55] David A Grossman and Ophir Frieder. *Information retrieval: Algorithms and heuristics*, volume 15. Springer Science & Business Media, 2012.

[56] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In *CIKM*, 2016.

[57] Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R. Bowman, and Noah A. Smith. Annotation artifacts in natural language inference data. In *NAACL*, 2018.

[58] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NIPS*, 2015.

[59] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

[60] Donna Harman. Overview of the fourth text retrieval conference (TREC-4). *NIST*, 1996.

[61] Donna K Harman. *Overview of the third text retrieval conference (TREC-3)*. DIANE Publishing, 1995.

[62] Helia Hashemi, Mohammad Aliannejadi, Hamed Zamani, and W. Bruce Croft. ANTIQUE: A non-factoid question answering benchmark. *CoRR*, abs/1905.08957, 2019.

[63] Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. Scalable modified Kneser-Ney language model estimation. In *ACL*, August 2013.

[64] Marti A. Hearst. Tilebars: Visualization of term distribution information in full text information access. In *CHI*, 1995.

[65] James M Heilman and Andrew G West. Wikipedia and medicine: quantifying readership, editors, and the significance of natural language. *Journal of medical Internet research*, 17(3), 2015.

[66] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *CoRR*, abs/1606.08415, 2016.

[67] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[68] Sebastian Hofstätter and Allan Hanbury. Let's measure run time! extending the IR replicability infrastructure to include performance aspects. In *OSIRRC@SIGIR*, 2019.

[69] Sebastian Hofstätter, Markus Zlabinger, and Allan Hanbury. TU Wien @ TREC deep learning '19 – simple contextualization for re-ranking. In *TREC*, 2019.

[70] Sebastian Hofstätter, Markus Zlabinger, and A. Hanbury. Interpretable and time-budget-constrained contextualization for re-ranking. In *ECAI*, 2020.

[71] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[72] Baotian Hu, Z. Lu, Hang Li, and Q. Chen. Convolutional neural network architectures for matching natural language sentences. In *NIPS*, 2014.

[73] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*, 2013.

[74] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1), 2017.

[75] Kai Hui, Andrew Yates, Klaus Berberich, and Gerard de Melo. PACRR: A position-aware neural ir model for relevance matching. In *EMNLP*, 2017.

[76] Kai Hui, Andrew Yates, Klaus Berberich, and Gerard de Melo. Co-PACRR: A context-aware neural ir model for ad-hoc retrieval. In *WSDM*, 2018.

[77] Samuel Huston and W Bruce Croft. Parameters learned in the comparison of retrieval models using term dependencies. *Technical Report*, 2014.

[78] Shiyu Ji, Jinjin Shao, and Tao Yang. Efficient Interaction-based Neural Ranking with Locality Sensitive Hashing. In *WWW*, 2019.

[79] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for Natural Language Understanding. *CoRR*, abs/1909.10351, 2019.

[80] Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's multilingual neural machine translation system: Enabling zero-shot translation. *TACL*, 2017.

[81] K Sparck Jones, Steve Walker, and Stephen E. Robertson. A probabilistic model of information retrieval: development and comparative experiments: Part 2. *Information processing & management*, 36(6), 2000.

[82] Amit D Kalaria and Ross W Filice. Comparison-bot: an automated preliminary-final report comparison system. *Journal of digital imaging*, 2015.

[83] Guolin Ke, Q. Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and T. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NIPS*, 2017.

[84] Omar Khattab and Matei Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *SIGIR*, 2020.

[85] Joo-Kyung Kim, Young-Bum Kim, Ruhi Sarikaya, and Eric Fosler-Lussier. Cross-lingual transfer learning for pos tagging without cross-lingual resources. In *EMNLP*, 2017.

[86] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[87] Bevan Koopman, Liam Cripwell, and Guido Zuccon. Generating clinical queries from patient narratives: A comparison between machines and humans. In *SIGIR*, 2017.

[88] Kui-Lam Kwok, Laszlo Grunfeld, H. L. Sun, and Peter Deng. TREC 2004 robust track experiments using PIRCS. In *TREC*, 2004.

[89] Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *CoRR*, abs/1901.07291, 2019.

[90] Zhen-Zhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *CoRR*, abs/1909.11942, 2019.

[91] Francesco Lettich, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Nicola Tonellotto, and Rossano Venturini. Parallel traversal of large ensembles of decision trees. *IEEE Transactions on Parallel and Distributed Systems*, page 14, 2018. ISSN 1045-9219. doi: 10.1109/TPDS.2018.2860982.

[92] Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *ACL*, volume 2, 2014.

[93] Bo Li, Ping Cheng, and Le Jia. Joint learning from labeled and unlabeled data for information retrieval. In *COLING*, 2018.

[94] Canjia Li, Yingfei Sun, Ben He, Le Wang, Kai Hui, Andrew Yates, Le Sun, and Jungang Xu. NPRF: A neural pseudo relevance feedback framework for ad-hoc information retrieval. In *EMNLP*, 2018.

[95] Canjia Li, Andrew Yates, Sean MacAvaney, Ben He, and Yingfei Sun. PARADE: Passage representation aggregation for document reranking. *arXiv*, abs/2008.09093, 2020. URL `https://arxiv.org/abs/2008.09093`.

[96] Nut Limsopatham, Richard McCreadie, M-Dyaa Albakour, Craig MacDonald, Rodrygo L. T. Santos, and Iadh Ounis. University of glasgow at TREC 2012: Experiments with terrier. In *TREC*, 2012.

[97] Jimmy Lin. The neural hype and comparisons against weak baselines. In *SIGIR Forum*, 2018.

[98] Jimmy Lin. The neural hype, justified! a recantation. In *SIGIR Forum*, 2019.

[99] Jimmy Lin and Peilin Yang. The Impact of Score Ties on Repeatability in Document Ranking. In *SIGIR*, 2019. arXiv: 1807.05798.

[100] Xinshi Lin and Wai Lam. CUIS team for TREC 2017 CAR track. In *TREC*, 2017.

[101] Robert Litschko, Goran Glavaš, Simone Paolo Ponzetto, and Ivan Vulić. Unsupervised cross-lingual information retrieval using monolingual data only. In *SIGIR*, 2018.

[102] Linqing Liu, Wei Yang, Jinfeng Rao, Raphael Tang, and Jimmy Lin. Incorporating contextual and syntactic structures improves semantic similarity modeling. In *EMNLP*, 2019.

[103] Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. Linguistic knowledge and transferability of contextual representations. In *NAACL*, 2019.

[104] Xitong Liu, Peilin Yang, and Hui Fang. Entity came to rescue - leveraging entities to minimize risks in web search. In *TREC*, 2014.

[105] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar S. Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke S. Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *CoRR*, abs/1907.11692, 2019.

[106] Daniel Loureiro, Kiamehr Rezaee, Mohammad Taher Pilehvar, and José Camacho-Collados. Language models and word sense disambiguation: An overview and analysis. *CoRR*, abs/2008.11608, 2020.

[107] Sean MacAvaney. OpenNIR: A complete neural ad-hoc ranking pipeline. In *Proceedings of the Thirteenth ACM International Conference on Web Search and Data Mining*, pages 845–848, 2020. doi: 10.1145/3336191.3371864.

[108] Sean MacAvaney, Kai Hui, and Andrew Yates. An approach for weakly-supervised deep information retrieval. In *SIGIR 2017 Workshop on Neural Information Retrieval*, 2017. URL `https://arxiv.org/abs/1707.00189v2`.

[109] Sean MacAvaney, Andrew Yates, Arman Cohan, Luca Soldaini, Kai Hui, Nazli Goharian, and Ophir Frieder. Characterizing question facets for complex answer retrieval. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1205–1208, 2018. doi: 10.1145/3209978.3210135. URL `https://arxiv.org/abs/1805.00791`.

[110] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. CEDR: Contextualized embeddings for document ranking. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1101–1104, 2019. doi: 10.1145/3331184.3331317. URL `https://arxiv.org/abs/1904.07094`.

[111] Sean MacAvaney, Andrew Yates, Arman Cohan, Luca Soldaini, Kai Hui, Nazli Goharian, and Ophir Frieder. Overcoming low-utility facets for complex answer retrieval. *Information Retrieval Journal*, 22:395–418, 2019. doi: 10.1007/s10791-018-9343-0. URL https://link.springer.com/article/10.1007/s10791-018-9343-0.

[112] Sean MacAvaney, Andrew Yates, Kai Hui, and Ophir Frieder. Content-based weak supervision for ad-hoc re-ranking. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 993–996, 2019. doi: 10.1145/3331184.3331316. URL https://arxiv.org/abs/1707.00189.

[113] Sean MacAvaney, Arman Cohan, and Nazli Goharian. SLEDGE-Z: A zero-shot baseline for COVID-19 literature search. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 2020. URL https://arxiv.org/abs/2010.05987.

[114] Sean MacAvaney, Arman Cohan, Nazli Goharian, and Ross Filice. Ranking significant discrepancies in clinical reports. In *Proceedings of the 42nd European Conference on Information Retrieval Research*, pages 238—-245, 2020. doi: 10.1007/978-3-030-45442-5_30. URL https://link.springer.com/chapter/10.1007/978-3-030-45442-5_30.

[115] Sean MacAvaney, Sergey Feldman, Nazli Goharian, Doug Downey, and Arman Cohan. ABNIRML: Analyzing the behavior of neural ir models. *arXiv*, abs/2011.00696, 2020. URL https://arxiv.org/abs/2011.00696.

[116] Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. Training curricula for open domain answer

re-ranking. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 529–538, 2020. doi: 10.1145/3397271.3401094. URL https://arxiv.org/abs/2004.14269.

[117] Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. Efficient document re-ranking for transformers by precomputing term representations. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 49–58, 2020. doi: 10.1145/3397271.3401093. URL https://arxiv.org/abs/2004.14255.

[118] Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. Expansion via prediction of importance with contextualization. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1573–1576, 2020. doi: 10.1145/3397271.3401262. URL https://arxiv.org/abs/2004.14245.

[119] Sean MacAvaney, Luca Soldaini, and Nazli Goharian. Teaching a new dog old tricks: Resurrecting multilingual retrieval using zero-shot learning. In *Proceedings of the 42nd European Conference on Information Retrieval Research*, pages 246–254, 2020. doi: 10.1007/978-3-030-45442-5_31. URL https://link.springer.com/chapter/10.1007/978-3-030-45442-5_31.

[120] Craig MacDonald and Nicola Tonellotto. Declarative experimentation in information retrieval using pyterrier. *ICTIR*, 2020.

[121] Martin A Makary and Michael Daniel. Medical error—the third leading cause of death in the us. *Bmj*, 353, 2016.

[122] Ramon Maldonado, Stuart Taylor, and Sanda M. Harabagiu. UTD HLTRI at TREC 2017: Complex answer retrieval track. In *TREC*, 2017.

[123] Irina Matveeva, Christopher J. C. Burges, Timo Burkard, Andy Laucius, and Leon Wong. High accuracy retrieval with multiple nested ranker. In *SIGIR*, 2006.

[124] R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *ACL*, 2019.

[125] Donald Metzler and W. Bruce Croft. A markov random field model for term dependencies. In *SIGIR*, 2005.

[126] Bhaskar Mitra and Nick Craswell. An updated duet model for passage re-ranking. *CoRR*, 2019.

[127] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. Learning to match using local and distributed representations of text for web search. In *WWW*, 2017.

[128] Bhaskar Mitra, Nick Craswell, et al. An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval*, 2018.

[129] Federico Nanni, Bhaskar Mitra, Matt Magnusson, and Laura Dietz. Benchmark for complex answer retrieval. In *ICTIR*, 2017.

[130] Courtney Napoles, Keisuke Sakaguchi, and Joel Tetreault. Jfleg: A fluency corpus and benchmark for grammatical error correction. In *EACL*, 2017.

[131] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization. In *EMNLP*, 2018.

[132] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. Ms marco: A human generated machine reading comprehension dataset. In *NIPS*, 2016.

[133] Maximilian Nickel, Lorenzo Rosasco, and Tomaso A. Poggio. Holographic embeddings of knowledge graphs. In *AAAI*, 2016.

[134] Rodrigo Nogueira and Kyunghyun Cho. Task-oriented query reformulation with reinforcement learning. In *EMNLP*, 2017.

[135] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with BERT. *CoRR*, abs/1901.04085, 2019.

[136] Rodrigo Nogueira and Jimmy Lin. From doc2query to doctttttquery, 2019. URL `https://cs.uwaterloo.ca/~jimmylin/publications/Nogueira_Lin_2019_docTTTTTquery.pdf`.

[137] Rodrigo Nogueira, Kyunghyun Cho, Urjitkumar Patel, and Vincent Chabot. New york university submission to TREC-CAR 2017. In *TREC*, 2017.

[138] Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. Document expansion by query prediction. *CoRR*, abs/1904.08375, 2019.

[139] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Document ranking with a pretrained sequence-to-sequence model. *CoRR*, abs/2003.06713, 2020.

[140] Douglas W Oard and Fredric C Gey. The TREC 2002 arabic/english clir track. In *TREC*, 2002.

[141] Kezban Dilek Onal, Ye Zhang, Ismail Sengor Altingovde, Md Mustafizur Rahman, Pinar Karagoz, Alex Braylan, Brandon Dang, Heng-Lu Chang, Henna

Kim, Quinten McNamara, et al. Neural information retrieval: At the end of the early years. *Information Retrieval Journal*, 2018.

[142] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. A survey of the usages of deep learning in natural language processing. *CoRR*, abs/1807.10854, 2018.

[143] Iadh Ounis, Giambattista Amati, Vassilis Plachouras, Ben He, Craig Macdonald, and Christina Lioma. Terrier: A High Performance and Scalable Information Retrieval Platform. In *Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006)*, 2006.

[144] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. In *WWW*, 1999.

[145] Wei Pan, Hao Dong, and Yike Guo. Dropneuron: Simplifying the structure of deep neural networks. *CoRR*, abs/1606.07326, 2016.

[146] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, and Xueqi Cheng. A study of matchpyramid models on ad-hoc retrieval. In *NeuIR workshop*, 2016.

[147] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. Text matching as image recognition. In *AAAI*, 2016.

[148] Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Jingfang Xu, and Xueqi Cheng. Deeprank: A new deep architecture for relevance ranking in information retrieval. In *CIKM*, 2017.

[149] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In *Proceedings of the 1st International Conference on Scalable Information Systems*, 2006.

[150] Gustavo Penha and Claudia Hauff. Curriculum learning strategies for ir: An empirical study on conversation response ranking. In *ECIR*, 2020.

[151] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

[152] Carol Peters, Martin Braschler, and Paul Clough. *Multilingual information retrieval: From research to practice*. Springer Science & Business Media, 2012.

[153] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *NAACL*, 2018.

[154] Jay Michael Ponte and W Bruce Croft. *A language modeling approach to information retrieval*. PhD thesis, University of Massachusetts at Amherst, 1998.

[155] Ye Qi, Devendra Singh Sachan, Matthieu Felix, Sarguna Padmanabhan, and Graham Neubig. When and why are pre-trained word embeddings useful for neural machine translation? In *NAACL*, 2018.

[156] Yifan Qiao, Chenyan Xiong, Zhenghao Liu, and Zhiyuan Liu. Understanding the behaviors of BERT in ranking. *CoRR*, abs/1904.07531, 2019.

[157] Meng Qu, Jian Tang, and Jiawei Han. Curriculum learning for heterogeneous star network embedding via deep reinforcement learning. In *WSDM*, 2018.

[158] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019. URL `https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf`.

[159] Colin Raffel, Noam Shazeer, Adam Kaleo Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

[160] Fiana Raiber and Oren Kurland. The technion at TREC 2013 web track: Cluster-based document retrieval. In *TREC*, 2013.

[161] Sudha Rao and J. Tetreault. Dear sir or madam, may i introduce the yafc corpus: Corpus, benchmarks and metrics for formality style transfer. In *NAACL*, 2018.

[162] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP*, 2019.

[163] D. Rennings, Felipe Moraes, and C. Hauff. An axiomatic approach to diagnosing neural ir models. In *ECIR*, 2019.

[164] Marco Túlio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of nlp models with checklist. In *ACL*, 2020.

[165] Kirk Roberts, Dina Demner-Fushman, Ellen M Voorhees, William R Hersh, Steven Bedrick, Alexander J Lazar, and Shubham Pant. Overview of the TREC 2017 precision medicine track. *NIST Special Publication*, 2017.

[166] Kirk Roberts, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, Kyle Lo, Ian Soboroff, Ellen Voorhees, Lucy Lu Wang, and William R Hersh. TREC-COVID: Rationale and Structure of an Information Retrieval Shared Task for COVID-19. *Journal of the American Medical Informatics Association*, 05 2020.

[167] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at TREC-3. *NIST Special Publication*, 109, 1995.

[168] Anna Rogers, O. Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works. *TACL*, 2020.

[169] Max Roser, Hannah Ritchie, and Esteban Ortiz-Ospina. Internet. https://ourworldindata.org/internet, 2019. Last accessed: 2019/09/15.

[170] Corby Rosset, Damien Jose, Gargi Ghosh, Bhaskar Mitra, and Saurabh Tiwary. Optimizing query evaluations using reinforcement learning for web search. In *SIGIR*, 2018.

[171] Alexander T Ruutiainen, Mary H Scanlon, and Jason N Itri. Identifying benchmarks for discrepancy rates in preliminary interpretations provided by radiology trainees at an academic institution. *Journal of the American College of Radiology*, 8(9), 2011.

[172] Mrinmaya Sachan and Eric P. Xing. Easy questions first? a case study on curriculum learning for question answering. In *ACL*, 2016.

[173] Tetsuya Sakai and Noriko Kando. On information retrieval metrics designed for evaluation with incomplete relevance assessments. *Information Retrieval*, 11(5), 2008.

[174] Shadi Saleh and Pavel Pecina. Term selection for query expansion in medical cross-lingual information retrieval. In *ECIR*, 2019.

[175] Mark Sanderson. Word sense disambiguation and information retrieval. In *SIGIR*, 1994.

[176] Evan Sandhaus. The new york times annotated corpus. *Linguistic Data Consortium, Philadelphia*, 6(12), 2008.

[177] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *Workshop on Energy Efficient Machine Learning and Cognitive Computing @ NeuIPS*, 2019.

[178] Jainisha Sankhavara. Biomedical document retrieval for clinical decision support system. In *ACL*, 2018.

[179] Shota Sasaki, Shuo Sun, Shigehiko Schamoni, Kevin Duh, and Kentaro Inui. Cross-lingual learning-to-rank with shared representations. In *NAACL*, 2018.

[180] Michael Schuhmacher, Laura Dietz, and Simone Paolo Ponzetto. Ranking entities for web queries through text and knowledge. In *CIKM*, 2015.

[181] Sebastian Schuster, Sonal Gupta, Rushin Shah, and Mike Lewis. Cross-lingual transfer learning for multilingual task oriented dialog. In *NAACL*, 2019.

[182] David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

[183] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *ACL*, 2017.

[184] Sanghyun Seo and Juntae Kim. Efficient weights quantization of convolutional neural networks using kernel density estimation based non-uniform quantizer. *Appl. Sci*, 2019.

[185] Sofia Serrano and Noah A. Smith. Is attention interpretable? In *ACL*, 2019.

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

[187] Amit Singh. Entity based Q&A retrieval. In *EMNLP*, 2012.

[188] Sameer Singh, Amarnag Subramanya, Fernando Pereira, and Andrew McCallum. Wikilinks: A large-scale cross-document coreference corpus labeled via links to wikipedia. *University of Massachusetts, Amherst, Tech. Rep. UM-CS-2012*, 15, 2012.

[189] Luca Soldaini, Andrew Yates, and Nazli Goharian. Denoising clinical notes for medical literature retrieval with convolutional neural model. In *CIKM*, 2017.

[190] Trevor Strohman, Donald Metzler, Howard Turtle, and W Bruce Croft. Indri: A language model-based search engine for complex queries. In *Proceedings of the International Conference on Intelligent Analysis*, 2005.

[191] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient Knowledge Distillation for BERT Model Compression. *CoRR*, abs/1908.09355, 2019.

[192] Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling task-specific knowledge from BERT into simple neural networks. *CoRR*, abs/1903.12136, 2019.

[193] Zhiwen Tang and Grace Hui Yang. Deeptilebars: Visualizing term distribution for neural information retrieval. In *AAAI*, 2019.

[194] Tao Tao and ChengXiang Zhai. An exploration of proximity measures in information retrieval. In *SIGIR*, 2007.

[195] Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In *ACL*, 2019.

[196] Huy Nguyen Tien, Minh Nguyen Le, Yamasaki Tomohiro, and Izuha Tatsuya. Sentence modeling via multiple word embeddings and multi-level comparison for semantic textual similarity. *Information Processing and Management*, 56, 2018.

[197] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. Efficient query processing for scalable web search. *Foundations and Trends in Information Retrieval*, 12 (4–5), 2018. ISSN 1554-0669.

[198] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

[199] Ellen Voorhees, D Harman, and R Wilkinson. The sixth text retrieval conference (TREC-6). In *TREC*, 1998.

[200] Ellen M Voorhees. Using WordNet to disambiguate word senses for text retrieval. In *SIGIR*, 1993.

[201] Ellen M. Voorhees. The TREC robust retrieval track. *SIGIR Forum*, 39, 2005.

[202] Ellen M Voorhees. Overview of the TREC 2005 Robust Retrieval Track. In *TREC*, 2005.

[203] Ellen M Voorhees and Donna Harman. Overview of the fifth text retrieval conference (TREC-5). In *TREC*, volume 97, 1996.

[204] Ivan Vulić and Marie-Francine Moens. Monolingual and cross-lingual information retrieval models based on (bilingual) word embeddings. In *SIGIR*, 2015.

[205] Jessica Walls, Natalie Hunter, Penelope MA Brasher, and Stephen GF Ho. The depictors study: discrepancies in preliminary interpretation of ct scans between on-call residents and staff. *Emergency radiology*, 16(4), 2009.

[206] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *SIGIR*, 2017.

[207] Lidan Wang, Jimmy Lin, and Donald Metzler. A cascade ranking model for efficient ranked retrieval. In *SIGIR*, 2011.

[208] Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Darrin Eide, Kathryn Funk, Rodney Kinney, Ziyang Liu, William. Merrill, Paul Mooney, Dewey A. Murdick, Devvret Rishi, Jerry Sheehan, Zhihong Shen, Brandon Stilson, Alex D. Wade, Kuansan Wang, Christopher Wilhelm, Boya Xie, Douglas M. Raymond, Daniel S. Weld, Oren Etzioni, and Sebastian Kohlmeier. CORD-19: The COVID-19 open research dataset. *ArXiv*, abs/2004.10706, 2020.

[209] Shuohang Wang, Sheng Zhang, Yelong Shen, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, and Jing Jiang. Unsupervised Deep Structured Semantic Models for Commonsense Reasoning. In *NAACL*, 2019.

[210] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, 2014.

[211] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019.

[212] Chenyan Xiong and Jamie Callan. Query expansion with Freebase. In *ICTIR*, 2015.

[213] Chenyan Xiong, James P. Callan, and Tie-Yan Liu. Word-entity duet representations for document ranking. In *SIGIR*, 2017.

[214] Chenyan Xiong, Zhuyun Dai, James P. Callan, Zhiyuan Liu, and Russell Power. End-to-end neural ad-hoc ranking with kernel pooling. In *SIGIR*, 2017.

[215] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, J. Liu, P. Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *CoRR*, abs/2007.00808, 2020.

[216] Chen Xu, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha. Alternating multi-bit quantization for recurrent neural networks. In *ICLR*, 2018.

[217] Peilin Yang, Hui Fang, and Jimmy Lin. Anserini: Enabling the use of lucene for information retrieval research. In *SIGIR*, 2017.

[218] Wei Yang, Kuang Lu, Peilin Yang, and Jimmy Lin. Critically examining the "neural hype": Weak baselines and the additivity of effectiveness gains from neural ranking models. In *SIGIR*, 2019.

[219] Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. End-to-end open-domain question answering with BERTserini. *CoRR*, abs/1901.04085, 2019.

[220] Wei Yang, Haotian Zhang, and Jimmy Lin. Simple applications of BERT for ad hoc document retrieval. *CoRR*, abs/1903.10972, 2019.

[221] Zhilin Yang, Ruslan Salakhutdinov, and William W Cohen. Transfer learning for sequence tagging with hierarchical recurrent networks. *CoRR*, abs/1703.06345, 2017.

[222] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, 2019.

[223] Andrew Yates and Nazli Goharian. ADRTrace: Detecting expected and unexpected adverse drug reactions from user reviews on social media sites. In *ECIR*, 2013.

[224] Andrew Yates and Kai Hui. DE-PACRR: Exploring layers inside the pacrr model. In *NeuIR*, 2017.

[225] Andrew Yates, Nazli Goharian, and Ophir Frieder. Relevance-ranked domain-specific synonym discovery. In *ECIR*, 2014.

[226] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *ACL*, 2015.

[227] Holly Young. The digital language divide. http://labs.theguardian.com/digital-language-divide/, 2015. Last accessed: 2019/09/15.

[228] Hamed Zamani, Mostafa Dehghani, W. Bruce Croft, Erik G. Learned-Miller, and Jaap Kamps. From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In *CIKM*, 2018.

[229] Yukun Zheng, Zhen Fan, Yiqun Liu, Cheng Luo, Min Zhang, and Shaoping Ma. Sogou-qcl: A new dataset with click relevance label. In *The 41st International*

ACM SIGIR Conference on Research & Development in Information Retrieval, 2018.