Generating Natural Language Queries for More Effective Ranking

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy.

Binsheng Liu

Master in Computer Software and Theory
Northeastern University, China

Bachelor in Software Engineering
Northeastern University, China

School of Computing Technologies
College of Science, Technology, Engineering and Maths
RMIT University

September, 2021

To the memory of my grandma,
He Hanyou,
who is no longer with me but whose love continues to support me.

To my parents,
Luo Zhengfeng and Liu Diandong,
who always make sacrifices to give me a better life.

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged; and, ethics procedures and guidelines have been followed.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Binsheng Liu
School of Computing Technologies
RMIT University, Melbourne, Australia
September 24, 2021

Acknowledgements

I started searching for an opportunity to do a PhD several years after completing my master's degree. Prof. Shane Culpepper trusted me and offered me a scholarship. My adventure would not have started without his support. During the PhD, Prof. Shane Culpepper, Prof. Oren Kurland and Dr. Xiaolu Lu supervised me with patience and kindness, inspired me to do good research, and supported me professionally and personally. I'm extremely grateful to them.

I would also like to extend my deepest gratitude to my collaborator Asst. Prof. Hamed Zamani. Discussions with him sharpened my thoughts and bettered my research.

I'm also grateful to my colleagues Joel Mackenzie, Luke Gallagher, and Rodger Benham. They have been a source of help and a source for understanding Australian culture. I'm also thankful to many other colleagues at RMIT who welcomed me and took time to chat with me. I have cherished those memories, especially during the lockdowns. Thanks also to the coauthors, mentors, academics, and students I met in various locations, who gave me suggestions and encouragement.

Finally, I would like to thank my friends and family who lent me emotional support. Anupriya, Aviraj, Elisa and Andres, Lily and Oleg, Mahshid, Nasrin, Ruda, Sanje, Sarigama, Satnam, Yunzhuang, and many others kept me company in a foreign country. Life would be much harder without them. My family always warms me up. Their love helped me through difficult times.

Contents

List of Tables

List of Figures

List of Symbols

Table 1: List of Symbols

Notation	Description
$D = \{d_1, d_2, \dots\}$	A document with terms d_1, d_2, etc. if not otherwise specified.
$Q = \{q_1, q_2, \dots\}$	A query with terms q_1, q_2, etc. if not otherwise specified.
$X = \{x_1, x_2, \dots\}$	Generally, a set X with elements x_1, x_2, etc.
$\mathbb{R}$	The set of real numbers.
$\lvert X \rvert$	The number of elements in X. For example $\lvert Q \rvert$ is query length.
$tf(D, x)$	Term frequency of term x in document d.
$df(x)$	Document frequency of term x.
$\mathbf{x}$	Lower case boldface for column vectors.
$\lVert \mathbf{x} \rVert$	Euclidean norm of vector $\mathbf{x}$. $\lVert \mathbf{x} \rVert = \sqrt{\sum_i x_i^2}$.
$\mathbf{X}$	Upper case boldface for matrices and high-dimensional tensors.
$\mathbf{x}^\top$, $\mathbf{X}^\top$	The transpose of $\mathbf{x}$ and $\mathbf{X}$.
$\underset{m \times n}{\mathbf{X}}$	An annotated matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$.
$z = \mathbf{x} \cdot \mathbf{y}$ $z = \mathbf{x}\mathbf{y}$	Dot product.
$\underset{m}{\mathbf{z}} = \underset{m \times n}{\mathbf{W}} \times \underset{n}{\mathbf{x}}$ $\mathbf{z} = \mathbf{W}\mathbf{x}$	Matrix-vector product.
$\underset{m \times n}{\mathbf{Z}} = \underset{m \times p}{\mathbf{X}} \times \underset{p \times n}{\mathbf{Y}}$ $\mathbf{Z} = \mathbf{X}\mathbf{Y}$	Matrix-matrix product.
$\underset{b \times m \times n}{\mathbf{Z}} = \underset{b \times m \times p}{\mathbf{X}} \times \underset{b \times p \times n}{\mathbf{Y}}$	Batched matrix-matrix product[1].

[1] https://pytorch.org/docs/stable/generated/torch.matmul.html

Abstract

Queries are formal representations of information needs and play a central role in information retrieval. Recent pretrained Transformer models have largely improved our abilities of processing natural language texts, including natural language queries. The way we understand queries, the techniques we use to generate queries need to be updated accordingly. This dissertation focuses on *queries*, particularly *query generations*.

Over the last 30 years researchers have been trying many ways to optimize queries. Relevance modeling is a classic technique which tries to surface more relevance information from pseudo-relevant documents. Our first attempt in query generation is incorporating field information into this technique. However, relevance models often result in long, uninterpretable queries. Our experiments also show that relevance models tend to make small improvements on web collections. We then perform an analysis on the value of rewriting queries, using automatically generated queries from search log and human-written queries from crowdsourcing. Our results show that rewriting queries can further improve retrieval effectiveness by a large amount. But, automatically generated queries are still not as good as human written queries, which motivates us to study *query generation* techniques by leveraging the recent progress in neural networks.

Query generation is a fundamental and versatile technique that has diverse applications. It can be used to produce query variations, reformulate queries, enrich documents, and so forth. Recent transformer networks has facilitated language generation tasks. These generation models are often trained with supervised learning, but we have observed that such learning objectives do not necessarily lead to effective queries, which means the generations have high readability but are often not effective as queries. We propose a novel task SNLQ (Strong Natural Language Queries) to combine readability and effectiveness objectives. To achieve both objectives, we propose a two-step approach — supervised learning for readability followed by reinforcement learning for effectiveness.

Finally, while we explore natural language query generation, the new form of queries has posed new challenges to ranking models compared to traditional keyword queries. Similar to generation models, ranking models also benefit from transformer networks. Inspired by traditional generative and discriminative approaches to ranking, we design a method to incorporate query generation (generative) into a ranking model (discriminative) so we can use a unified transformer architecture to transfer knowledge between query generation and ranking which results in more generalized ranking models.

1 Introduction

Search engines process millions or even billions of *queries* every single day. Many of these queries come from users who have diverse *information needs*. Unsurprisingly, queries can vary dramatically from simple *keyword* queries such as "facebook log in" to complex *natural language* queries such as "who was the us president when the red sox won the world series". Modern search engines may use user profile data such as location and search history to aid the search, but the query is still the most important and sometimes the only information a search engine has to understand the user's intent. Understanding the query and the information need it represents has always been the central task to a search engine.

In the early days, our search engines were limited to treating a query as a few keywords which contains only the most salient words but disregard word ordering. For decades, users have also been accustomed to using such keyword queries to express their information needs. Many of us might have learned to not use *stopwords* such as "a", "an", and "the" in the query as they would be discarded by the search engine. If we want to search for the first President of the United States, we can use a query like "first president united states" instead of "~~who is the~~ first president ~~of the~~ united states". We might also have learned that word ordering did not matter so "united states first president" is also likely to work. While users try their best to tailor the queries in the keyword language our search engines prefer, common problems in human languages such as imprecision, vagueness and ambiguity still happen.

The simplest example of query problems is perhaps spelling mistakes. Studies have found 10% [48] to 20% [63] queries contain spelling mistakes. This problem is relatively easy to fix, and we have mature techniques to fix them. Figure 1.1 shows an example of how a search engine can fix spelling errors and provide query suggestions. Spelling mistakes can be detected by looking up a dictionary and the correct spellings can be suggested through a properly designed user interface. With such an interface, search engines can also suggest related queries ("Britney Griner" and "Britney Theriot" in Figure 1.1) to prevent such errors from happening as the user keeps typing.

These front-end optimizations are helpful in improving query quality, but they do not solve the deeper problems in expressing information needs unambiguously. An information need can be represented by different terms, which is known as *synonymy*. The query "apple company"

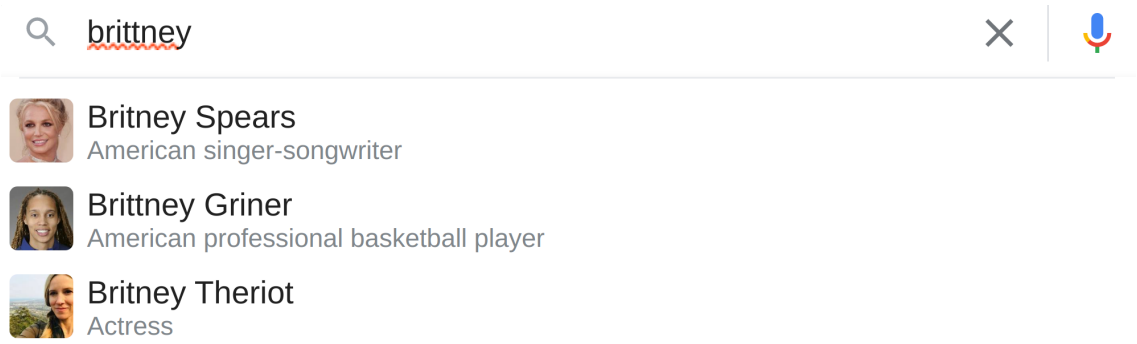


Figure 1.1: Spell correction and query suggestion interface. Spell mistakes can be identified and corrected before a query is issued to a search engine.

and "apple corporation" refer to the same thing; "teen pregnancy" and "teenage pregnancy" also refer to the same topic. These phrases rarely cause trouble to humans but can result in substantial problems known as *vocabulary mismatches* for search engines. The vocabulary mismatch problem happens when *exact matching* is applied in retrieving documents where terms in a query and a document are expected to be exactly the same. A document using "teen pregnancy" may be simply missed for the query "teenage pregnancy" because "teen" and "teenage" do not match.

The translation from an information need to a query also relies on a user's cognitive abilities, resulting in queries with varying quality. Belkin [12] has referred to users' needs for information as the *anomalous state of knowledge* where users have a gut feeling but do not know exactly what they want. Imagine a scenario where a user is trying to look for information about this phenomenon. Without knowing the terminology in advance, they will probably end up with several attempts in formulating the query until they eventually learned the phrase "anomalous state of knowledge" from related documents. These are just a few examples of challenges we are faced with in processing queries. All these factors are hidden barriers between a user and search engine quality.

Although a human cannot guess the vocabulary of a collection and formulate the perfect query, we can learn to improve it though thanks to recent advances. Researchers have invented several techniques (e.g. relevance feedback [174], BM25-based query expansion [171] and relevance models [106]) to optimize a query. In the background, a query may be rewritten or expanded. Figure 1.2 shows the query "teen pregnancy" and the document containing "teenage pregnancy" are successfully matched. This type of vocabulary mismatches are relatively easy to identify. *Relevance models*, which were invented in 2001, are perhaps one of the most popular and obvious solutions for this problem. Just like a user struggling with the anomalous state of knowledge, relevance models improve a query by learning new words from top-ranked documents. Promising new words are added to formulate a new query. However, the outcome is often mixed. It is not unusual for relevance models to improve some queries but harm others [69], and

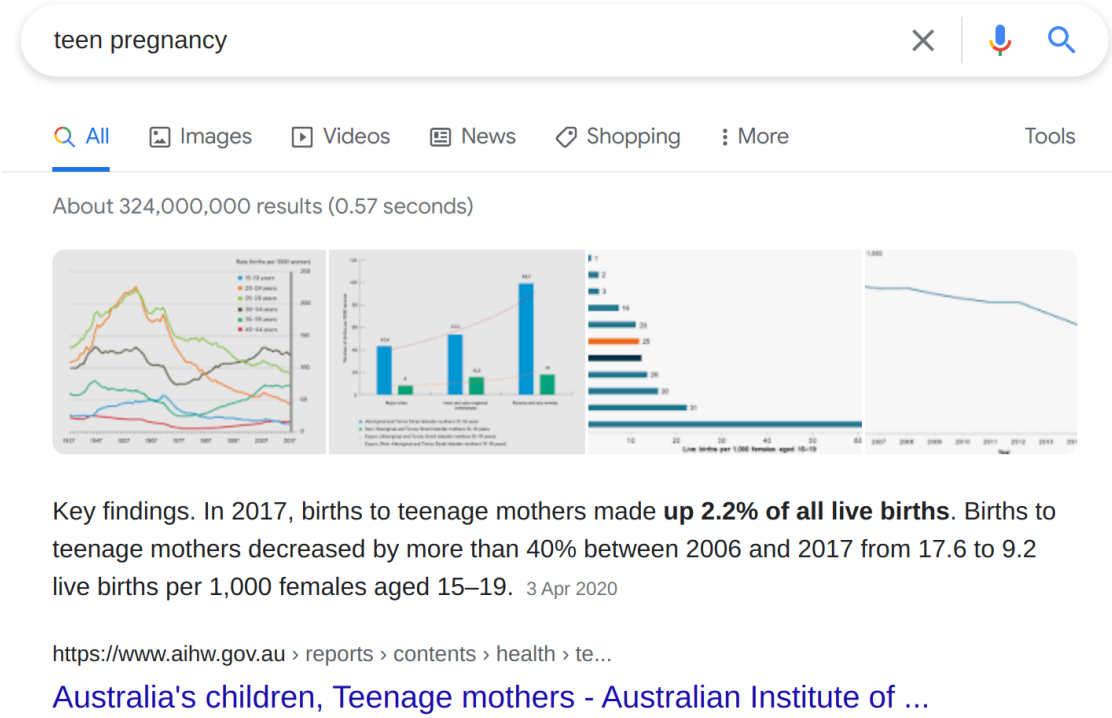


Figure 1.2: The query uses "*teen*" while the returned document uses "*teenage*". Vocabulary mismatches can be alleviated using query optimization techniques including pseudo-relevance feedback, synonyms, and word embeddings.

often, relevance models tend to work less effectively for collections built from web documents such as ClueWeb12B [123].

Commercial search engines have a different angle when rewriting queries. A commercial search engines processes a huge number of queries every day and these queries are a valuable source to be leveraged. A popular approach is to generate new queries by linking queries through the common documents clicked by users [43]. These new queries can form the candidates for query suggestion or work as a surrogate of the original query. Mining query logs has been a useful approach as we will shortly see that some generated queries can greatly improve retrieval performance by replacing the original queries. However, our analysis also shows that automatic generation techniques such as relevance models and query log based approaches still cannot match human beings in formulating queries. So, we still need more powerful tools for generating queries.

Recently, natural language queries have attracted more attention as techniques like voice search and digital assistants become popular. Keyword queries have intrinsic limitations in expressing an information need due to the lack of language structure. They are obviously not sufficient when complex phrases or sentences are needed to articulate the information needs. The

use of keyword queries is also not understood by users as an unnatural way of asking questions. In contrast, natural language queries tend to be more precise and less ambiguous, but they also pose fundamental challenges for search.

Figure 1.3: Natural language queries are more specific in representing the information need but are also more challenging for search engines to correctly understand.

How do retrieval models work for natural language queries? Until recently, our retrieval methods do not work very well with natural language queries which contain many common terms. In TREC[1] exercises, we see that short keyword title queries often outperform long natural description queries [77]. Figure 1.3 shows an example of a popular search engine failing to understand the meaning of a natural language query. Even if a user can precisely dictate their information need using natural languages, we are probably not in the position to fully take advantage of that in search.

Fortunately, recent progress in deep learning and natural language processing (NLP) has improved our ability to process natural language. Deep neural networks with rich linguistic abilities are widely adopted in various aspects of a search system. Over the last few years, using deep learning retrieval models to maximize effectiveness has become an integral part of a search engine. With these advances, we are also in a better position to explore more challenging tech-

[1]Text REtrieval Conference (TREC) is an annual conference where researchers trial their ideas using standard queries and document collections.

niques which was rarely explored before such as generating natural language queries, as existing query optimization techniques are mainly driven by keyword queries. Natural language query generation however can be a fundamental tool for extending the optimization ideas onto natural language queries. It can be used for query rewriting, query expansion, query simulation, query clarification, or conversational search.

Keyword queries are still the norm nowadays, but changes are happening. In this thesis, we start with a traditional keyword query optimization technique – relevance modeling, and identify the shortcomings of existing query generation techniques in Chapter 3. Then we explore generating effective natural language queries in Chapter 4. Finally, we combine ranking and query generation under one framework to maximize retrieval effectiveness in Chapter 5.

1.1 Organization

The rest of the thesis are organized as follows.

Chapter 2. Chapter 2 introduces the context and technical background of this thesis. We will first situate our work in the broader context of information retrieval research, and then develop related techniques in the rest of the thesis, including retrieval models, neural networks particularly sequence models, and reinforcement learning.

Chapter 3. Chapter 3 explores improving relevance modeling by incorporating field information. In particular, traditional relevance models do not use field information which is pervasive in web documents. We show that incorporating field information can improve retrieval effectiveness. While relevance models are popular, the resulted queries are often long and uninterpretable, which motivates our study in new query generation techniques. More motivation is provided through a comparative analysis identifying the performance gap between existing automatically generated queries and human written queries. The chapter focuses on the following **research question**: How can we improve relevance models, and what gaps remain in existing query generation techniques?

The content of the chapter appeared in the following publications:

- ✦ Binsheng Liu, X. Lu, O. Kurland, and J. S. Culpepper. Improving Search Effectiveness with Field-based Relevance Modeling. In Proc. ADCS, pages 1–4, 2018.
- ✦ Binsheng Liu, N. Craswell, X. Lu, O. Kurland, and J. S. Culpepper. A Comparative Analysis of Human and Automatic Query Variants. In Proc. ICTIR, pages 47–50, 2019.

Chapter 4. Chapter 4 explores generating effective queries that are readable. The NLP community has a similar task called *abstractive summarization* which has an emphasis on readability. The IR community has also explored generation queries but often only focuses on the effectiveness of generated queries. We propose a novel query generation task which combines readability

and effectiveness, and study how to achieve the best trade-off of both objectives. This chapter studies the following **research question**: How can we generate readable and effective queries?

The content of this chapter appeared in the following publications:

- Binsheng Liu, X. Lu, and J. S. Culpepper. Strong Natural Language Query Generation. Inf Retrieval J., 2021.

Chapter 5. Chapter 5 explores how to model ranking and query generation simultaneously. The idea traces back to traditional discriminative retrieval models and generative retrieval models. Both approaches can surface strong relevance signals, so we combine them in a multi-task learning framework. We implement the framework using transformers and show that retrieval effectiveness can substantially benefit from a joint modeling approach. This chapter studies the following **research question**: How can we jointly model ranking and query generation?

The content of the chapter appeared in the following publications:

- Binsheng Liu, H. Zamani, X. Lu, and J. S. Culpepper. Generalizing Discriminative Retrieval Models using Generative Tasks. In Proc. WWW, pages 3745–3756, 2021.

Chapter 6. Chapter 6 concludes the thesis, summarizes our contributions, and discusses open questions and future work.

2 Background

This thesis studies *queries* from multiple perspectives: optimizing queries, generating queries, and using queries to enrich ranking models. Section 2.1 reviews information retrieval in general including indexing, query representation, query optimization, ranking, and evaluation to provide a general context for our work. Sec 2.2 and Sec 2.3 introduces traditional retrieval models and neural ranking models which are the important underlying techniques for our work. They are also widely used as baselines in literature and serve as theoretical basis of our work. Sec 2.4 introduces sequence models with an emphasis on Transformers which are the background of Chapter 4 and Chapter 5. And finally, Section 2.5 introduces reinforcement learning which is another supporting machine learning technique.

2.1 Information Retrieval Overview

Generally speaking, information retrieval, as the name suggests, is mainly concerned with finding information. Nowadays, it has become an infrastructure of our digital life. From general information retrieval such as web search and library search, to more personalized retrieval such as email search and desktop search, and then to domain-specific legal information search and medical search, all the variety of tasks are supported by similar information retrieval techniques with an emphasis on different aspects. In literature, information retrieval has been defined in various wordings. Gerard Salton, in 1968, defined information retrieval as *"a field concerned with the structure, analysis, organization, storage, searching, and retrieval information"* [179] which is still valid for modern information retrieval after decades of development.

An information retrieval system consists of multiple components. Before it is able to process user queries, it indexes a large collection of candidate documents to support efficient access to the documents. We will briefly have a look at the basics of indexing and understand how queries are matched and how vocabulary mismatches happen in practice. After the index is built, a search begins with a user's information need. The user then translates the information need to a query which may or may not precisely represent the information need. In order to improve the quality of the query, retrieval systems can provide query suggestions or fix misspells for the user during the interactions. We would like the query to be as high quality as possible as it is the most important information we can access to understand the information need. Then the query is

passed down to a *retrieval model* which retrieves and ranks documents for the query. A retrieval model is at the core of a retrieval system as estimating relevance is even hard for human beings. We have observed a lot of disagreements in the TREC's relevance judging exercises where human beings are asked to decide query-document relevance. So, researchers have put a lot of effort into developing retrieval models in order to accurately estimate document relevance. The state-of-the-art retrieval systems often use multiple retrieval models. Throughout the retrieval pipeline, the models become more complex and resource-intensive and the pool of candidate documents also become smaller. After the retrieval stage, the retrieved documents are presented to users through a search engine results page (SERP).

We have put the main work of this thesis in the context of this workflow in Figure 2.1. The figure is also tailored to highlight the emphasis revolving around *queries* which are the core of this thesis. Chapter 3 studies query optimization and compares automatically generated queries and human formulated queries; motivated by the findings that completely rewriting queries can often lead to significant improvements, we focus our study on generating effective queries in Chapter 4; finally, Chapter 5 further leverages query generation signals by incorporating query generation into ranking to build a unified model.

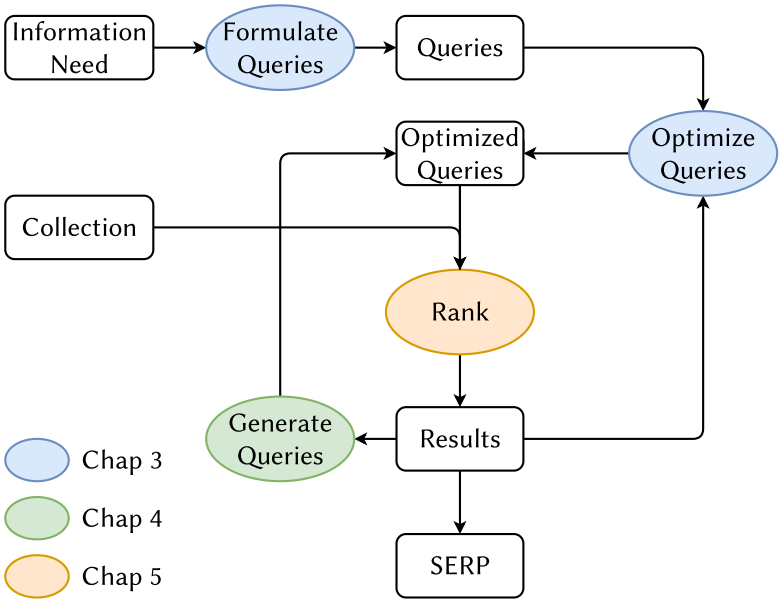


Figure 2.1: Information Retrieval overview.

2.1.1 Indexing

A large collection of documents needs to be properly indexed so they can be efficiently retrieved. *Inverted indexes* are the most commonly used indexing technique. Documents are parsed and

organized by the *terms* they contain, and the index can be considered as a dictionary where terms are the keys and document identifiers are the values. For example in Figure 2.2, doc 1 and 2 both contain the words "teenage", "teen", and "pregnancy", so they appear in the list of these three terms. Doc 4 contains "teen" and "pregnancy" so it only appears under these two terms. This structure allows efficient keyword matching. When a query is being processed, we can efficiently retrieve only documents that contain query terms and avoid examining the entire collection. However, if the query is "teenage pregnancy", doc 4, 5 and 6 which contains "teen" will be missed.

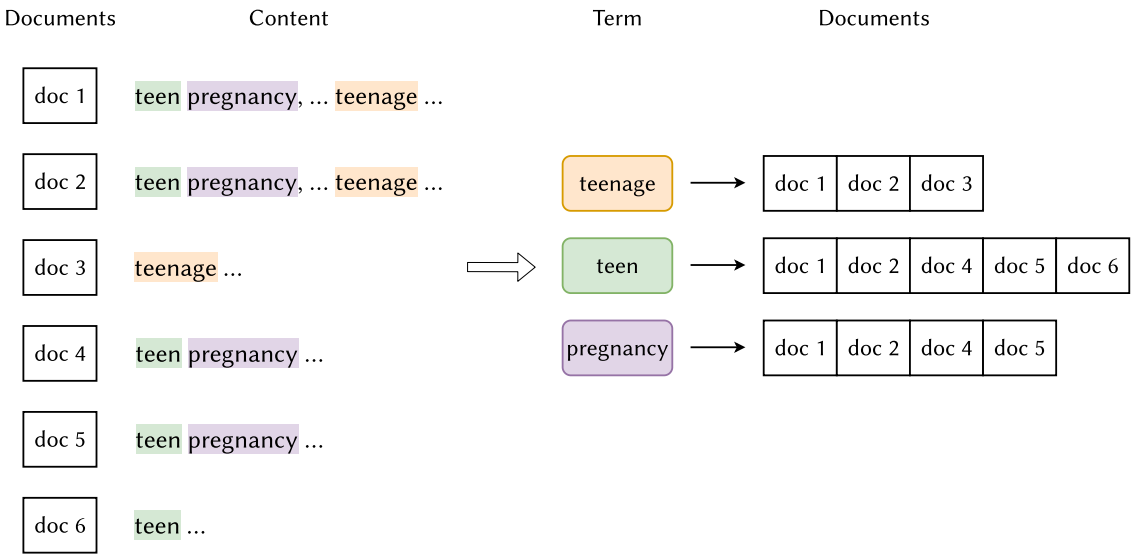


Figure 2.2: Inverted indexes that support keyword-based retrieval.

Inverted indexes play an important role in retrieving efficiency. Given a query, documents that do not contain any query term can be efficiently pruned and only a relatively small number of documents need to be scored. Now, inverted indexes are mostly involved in the early stages of a multi-stage retrieval system due to its efficiency.

2.1.2 Query Optimization

For a search engine, a search begins with a user query. However, the query may not be optimal as it can be easily misspelled or under-specify the user's information needs. To improve the quality of queries, Researchers have devised many techniques. We refer to techniques that try to improve the quality of a query as *query optimization* techniques.

Query optimizations can happen at two places as shown in Figure 2.1. The first place is from the original query where search engine can fix spelling mistakes and expand with synonyms by leveraging lexicon or query logs. The second place happens in a retrieve-optimize loop. The search engines optimize the query after retrieving some documents and use those retrieved doc-

uments to further optimize the query. This second optimization can be explicit where the user is asked to provide feedback or implicit where the search engine only uses the retrieved documents.

Spell Correction. Various studies show that 10% [48] to 20% [63] web queries contain spelling errors. For example, a Google report [1] also revealed that the query "britney spears" may be misspelled as "brittany spears", "brittney spears", "britany spears", and so on. Many mature algorithms exist for spelling corrections. These misspelled queries need to be fixed before they are used for retrieval. The most basic approach might be to compare out-of-vocabulary words with existing words by *edit distance* and replace them with the words having the smallest distance.

More recently, neural language models (we will discuss in Sec 2.4.3) are used for spell correction. Hu et al. [83] cast spelling correction as a mask language modeling problem. A misspelled word is masked out, and the language model is used to predict the missing word from the context. On a slightly different task of grammar correction, Katsumata and Komachi [95] leveraged the same idea but used an encoder-decoder model to recover the correct grammar.

Query Expansion. Query expansion aims at mitigating the mismatch between a query and a document by adding words into a query. The expansion can be done semi-automatically or automatically. Semi-automatic query expansion requires user interactions. We show an interface of semi-automatic query expansion in Figure 2.3. When "information" is typed in the search box, multiple expansion candidates are shown for the user to choose. The candidates can be generated by matching the already typed text against millions of queries in a large query logs. Automatic query expansion is often associated with pseudo-relevance feedback where a set of initial retrieval documents are used. We also refer to the set of documents as *pseudo-relevant* documents since they are *assumed* to be relevant in many query expansion techniques.

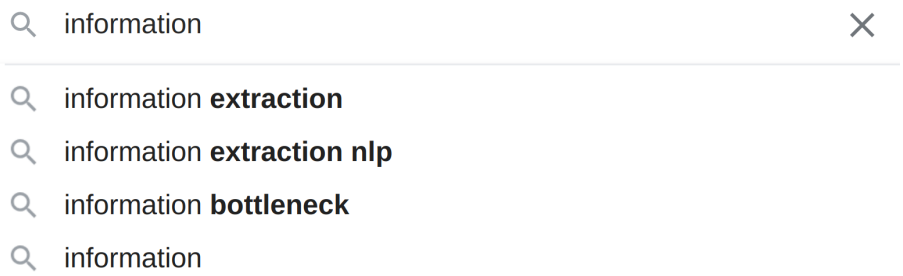


Figure 2.3: Semi-automatic query expansion from a commercial search engine.

Figure 2.4 shows a typical framework for automatic query expansion which is adopted by several previous work. First, an initial set of documents are retrieved using a traditional retrieval method such as BM25 or query likelihood. Second, some candidates are selected from the initial documents. The candidates for example can be individual terms, phrases, or fixed-windowed

[1] https://archive.google.com/jobs/britney.html

chunks. The candidates are then scored using different methods. The scored candidates can be used in multiple ways: top candidates can be added to the original query, all the candidates are used as a weighted query, or terms are sampled based on a probability. Either traditional relevance modeling [106], neural relevance modeling[137] or more complex neural network based expansion [226] fall into this framework.

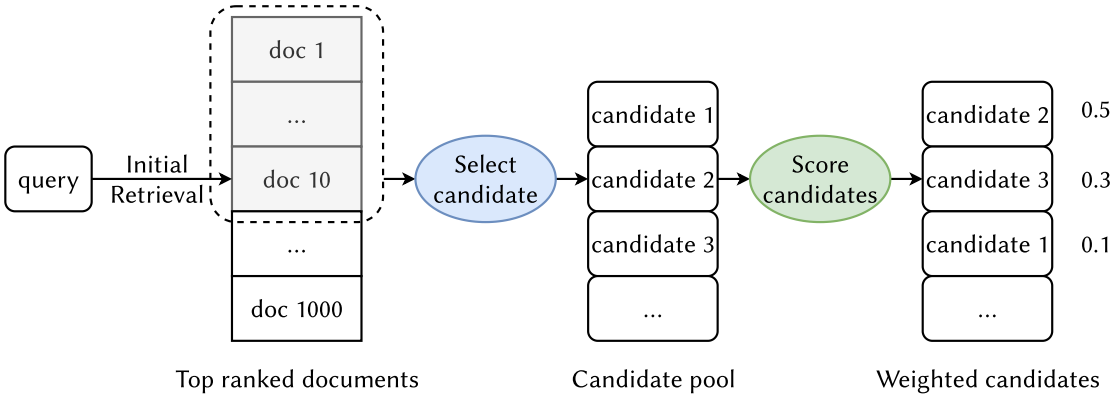


Figure 2.4: A typical automatic query expansion framework based on pseudo-relevant feedback.

2.1.3 Representations of Information

The distinction between a query and the underlying information needs represented by the query has been an essential component of Information Retrieval research for many years. Most IR researchers follow TREC test collection paradigm of using one single query to represent an information need, while others argue that using multiple representations or queries is superior to using a single query. In Table 2.1, we summarize the literature that exploits multiple representations of an information need. As the table suggests, in addition to query variations, there are other sources that can be leveraged as additional evidence during the retrieval process. We categorize the literature into four groups in terms of the source of evidence they discuss: Q (Query) means using multiple queries for an information need; D (Document) means using document title, abstract, body, and so on when ranking the document; S (System) means using different retrieval methods for the same retrieval task, e.g. query likelihood and BM25. Other than these representations of an information need, Robertson [169] also suggest the relationship, or relevance judgments, between queries and documents. Previous literature also has a different focus on justifying the use of multiple representations which we can mainly classify into three perspectives: the query variability (QV), the impact on system performance (SP), and the support from theoretical frameworks (TF). Respectively, some work follows the direction of query variability, and focus on how users formulate their information needs according to their cognitive differences; studies on system performance focus on the effectiveness benefits of using multi-

Table 2.1: Literature on information representations. Q: Query; D: Document; R: Relationship between query and document; S: System. QV: Query Variability; SP: System Performance; TF: Theoretical Framework.

Literature	Year	Source				Perspective		
		Q	D	R	S	QV	SP	TF
The Probability Ranking Principle in IR [169]	1977	✓	✓	✓				✓
An Evaluation of Factors Affecting Document Ranking by Information Retrieval Systems [126]	1979	✓			✓		✓	
A Study of the Overlap Among Document Representations [96]	1983		✓		✓		✓	
A Study of Information Seeking and Retrieving. III. Searchers, Searches, and Overlap [182]	1988	✓				✓		
Evaluation of an Inference Network-Based Retrieval Model [199]	1991	✓	✓					✓
The Effect of Multiple Query Representations on Information Retrieval System Performance [13]	1993	✓					✓	
Combination of Multiple Searches [186]	1994	✓			✓		✓	
Combining the Evidence of Multiple Query Representations for Information Retrieval [14]	1995	✓					✓	
Analyses of Multiple Evidence Combination [108]	1997	✓					✓	
Assessing the Cognitive Complexity of Information Needs [135]	2014	✓				✓		
User Variability and IR System Evaluation [8]	2015	✓				✓	✓	
UQV100: A Test Collection with Query Variability [9]	2016	✓				✓		
RMIT at the TREC CORE Track [19]	2018	✓			✓	✓	✓	

ple representations for retrieval; and finally theoretical frameworks focus on categorizing our understanding.

From a theoretical perspective, Robertson [169] provided a probabilistic framework which supports using multiple sources of evidence to improve the retrieval accuracy. They incorporate multiple queries, documents, query-document relationships to improve the retrieval in their framework. Turtle and Croft [199], using an inference network, justifies the use of multiple query variations, and they also suggest integrating NLP techniques.

However, most researchers especially the early ones focus on finding the relationship between query variations and system performance. Following Robertson [169], McGill [126] discover that different techniques or queries find very different documents. Katzer et al. [96] observe the same effect using multiple representations of documents. Furthermore, Belkin et al. [13] argued that an information need is so complex that any single representation is not sufficient, and

multiple queries or techniques capture different aspects of the information need. Their experiment shows that the performance improvement is positively correlated to the number of query representations. Another reason why multiple representation should be preferred comes from Lee [108]. Although there is some empirical evidence advocating the superiority of multiple representations, Lee [108] was the first to find that different ranked lists contain similar relevant documents but different non-relevant documents.

Before Shaw and Fox [186], the combination of multiple queries were usually performed before retrieval: multiple queries are first combined into a single query, for example using Boolean operators. Shaw and Fox [186] propose CombSUM fusion method which combines the retrieval lists of each query variation. The method adds up different scores from retrieval lists and works better than choosing any single score. There are also two variations of this method, CombMNZ and CombANZ [14]. The former multiplies the aggregated score with the number of lists in which the document appears, while the latter divides the score by the number. Vogt and Cottrell [202] extended the CombSUM method by assigning weights to retrieval lists.

Regarding human cognition, it's not surprising that query variability is strongly related to human cognitive skills. To date, most query variations are collected from human and referred to as user query variations. Saracevic and Kantor [182] studies multiple representations of an information need from a user perspective. Their results show that query variability may come from the difference in searchers' language skills and abstract thinking abilities. Bailey et al. [8] conducted an experiment in which TREC topics were categorized into Remember, Understand, and Analyse, according to their cognitive complexity. They then asked annotators to decide which topic belongs to which category. They find the Remember category has the highest agreement while there is not a clear boundary between Understand and Analyse. Following this, Bailey et al. [9] collected query variations for each topic from the TREC 2013 and 2014 Web Tracks. They wrote backstories for the 100 topics and collected variations through crowdsourcing systems. Crowd workers saw the backstories and wrote queries based on their understanding to the backstories. Using a similar approach, Benham et al. [19] collected variations for 250 topics of Robust04. In contrast to Bailey et al. [9], the people who created Robust04 variants are IR domain experts, and thus it's not surprising these variants have better quality than ClueWeb12B.

Although user query variations improve retrieval performance, they can't be easily deployed in a production system as they require human curation. Thus, automatic query generation is of special interest, but surprisingly there is little prior work on this topic. Sheldon et al. [187] proposed a process to induce multiple query variations from a starting query or description using the random walk model originally described by Craswell and Szummer [43]. Benham et al. [18] use a sampling based technique to automate the process of generating query variations. They induce relevance models using the title query, and sample query terms from the relevance models several times. These are the only prior work to report performance improvement with automatically generated queries, but to date, it is still unclear how the automatic query variations compare to user query variations and how we can generate effective queries.

2.1.4 Multi-Stage Retrieval

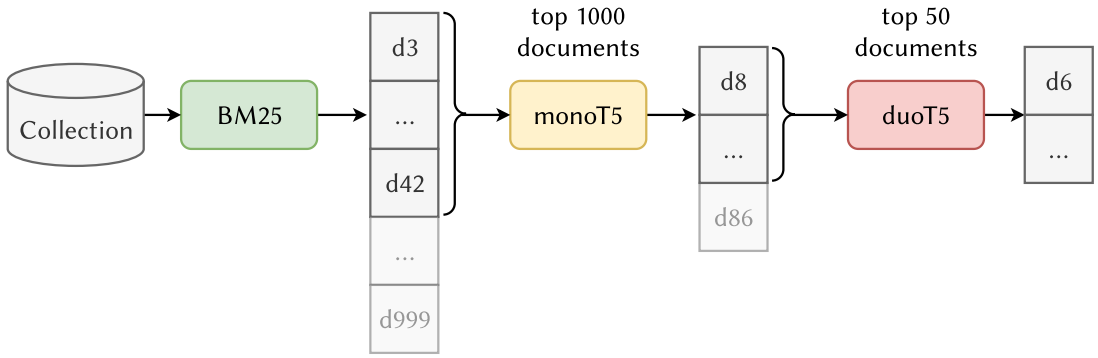


Figure 2.5: Multi-stage retrieval from Pradeep et al. [161]. The early precision of the ranking is improved after each stage as the computations get more and more expensive towards the end of the pipeline.

Modern search systems often consist of multiple stages of ranking to balance efficiency and effectiveness of different retrieval models. Such systems are also referred to as *cascading* systems. In the early stages of a cascade system, fast retrieval methods are often used to select some candidate documents from the entire collection. In the following stages, the candidates will be further reranked by more complex learning-to-rank or neural ranking models. As the ranking algorithms become more expensive, the candidate pool is gradually reduced to balance efficiency and effectiveness cost. Figure 2.5 illustrates a multi-stage ranking architecture proposed by Pradeep et al. [161]. In the first stage, they used BM25 to retrieve 1000 documents. In the second stage, they used a pointwise neural ranking model monoT5 to rank the 1000 documents. In the third stage, the top 50 documents from the second stage are grouped into pairs and fed into a more complex ranking model called duoT5. Each stage has a different focus in the pipeline. The early stages often focus on efficiency and high-recall which means to include as many relevant documents in the candidate pool as possible. The later stages often aim for high-precision as they are finely ranking the top documents.

The models used in different stages are often trained independently which may result in sub-optimal outcome. Gallagher et al. [70] have studied jointly training multiple ranking models by incorporating feature extraction cost into training, to achieve the best trade-off between efficiency and effectiveness albeit with a feature-based model and Gradient Boosted Regression Trees.

2.1.5 Evaluation of Retrieval

Evaluation is another well-studied area in IR. We evaluate our work in this thesis on reusable test collections with offline evaluation metrics. A test collection is composed with documents, queries and relevance judgments for query-document pairs. A list of documents returned by a retrieval system can be evaluated according to the judgments and the quality can be measured with various metrics. In this section we list the collections and metrics we used.

Datasets. Many test collections are released by Text REtrieval Conference (TREC) of National Institute of Standards and Technology[2]. Specifically, we use the following test collections:

- TREC Robust04 [204]: A test collection with about 528 thousand documents and 250 topics released in 2004.
- TREC ClueWeb09B [36]: A test collection of about 50 million documents and 200 topics released in 2009.
- TREC ClueWeb12B [37]: A test collection of about 52 million documents and 100 topics released in 2012.
- MS-MARCO [143]: A test collection of about 8 million documents and more than 500 thousand topics. Most topics contain only one "relevant" judgment.
- CAsT 2019 [54]: The document collection is built by concatenating MS-MARCO and TREC CAR [60]. There are 173 queries in this test collection.

We will discuss collection details when they are first used in the thesis.

Evaluation Metrics. Retrieval systems return a list of documents. Given the relevant documents R, we can represent the retrieval list as a relevance list. Let

$$L = \{r_i \in \{0, 1\} \,|i = 1, 2, \ldots, d\}$$

denote such a list where $r_i = 0$ indicates the ith document is non-relevant or not judged, and $r_i = 1$ indicates the ith document is relevant. A metric is defined over such a list. More generally, the metric can be defined at a cutoff k where only the first k documents are considered.

The first metric we consider is precision. It measures how many relevant documents are retrieved among all the retrieved documents.

$$Precision@k = \frac{\sum_i^k r_i}{k} \tag{2.1}$$

$P@k$ is commonly used an abbreviation for $Precision@k$. A problem of precision is when $k > |R|$ it never reaches 1.0. So a relatively small k such as $k = 10$ is more reliable in practice [134].

[2] https://trec.nist.gov

Recall is defined as a ratio of retrieved relevant documents over all relevant documents. It is particularly useful in a cascade retrieval system where early stages aim for high recall.

$$Recall@k = \frac{\sum_i^k r_i}{|R|} \tag{2.2}$$

Reciprocal rank cares only about the first retrieved relevant document. It is defined as the reciprocal of that document's rank position:

$$RR@k = \frac{1}{i}, \quad i = \min(\{i = 1, 2, \ldots, k | r_i > 0\}) \tag{2.3}$$

where i is the rank position of the first relevant document. The average reciprocal rank of a set of queries is called mean reciprocal rank (MRR). MRR may be more suitable when relevance judgments are shallow. For example, MS-MARCO has only 1 judgments for most documents and MRR is used as the official metric for their leaderboard. However, it is also considered unstable as a high RR query can compensate many low RR queries [134].

Average precision (AP) combines recall and precision and is defined as:

$$AP@k = \frac{1}{|R|} \sum_i^k r_i \cdot P@i \tag{2.4}$$

The average of this value over a set of queries is called mean average precision (MAP).

Discounted cumulative gain (DCG) is defined over graded judgments. That is, $r_i \in \{0, 1, 2, \ldots\}$ is not limited to binary values. DCG awards more penalties to ranking relevant documents at a lower position, so the *gain* of a relevant document is discounted as its rank position gets lower.

$$DCG@k = \sum_i^k \frac{r_i}{\log_2(i+1)} \tag{2.5}$$

DCG is often normalized in order to be compared among queries. The normalization factor is the DCG score of a perfectly ranked list, denoted as IDCG

$$NDCG@k = \frac{DCG@k}{IDCG@k} \tag{2.6}$$

so that NDCG is bounded by 0 and 1.

Rank biased precision (RBP) is based on the assumption that, after examining some documents, there is a probability p that the user will continue inspecting the rest of the documents. Patient users have a high p while impatient users have a low p. It is formulated as:

$$RBP@k = (1-p) \cdot \sum_i^k r_i \cdot p^{i-1} \tag{2.7}$$

2.2 Traditional Ranking Models

Judging relevance is a cognitive process which involves in understanding complex languages. Throughout the development of information retrieval, researchers have proposed many retrieval models hoping to accurately represent information and estimate the relevance to users' needs.

Strictly speaking, *retrieval* and *ranking* are different. The former refers to retrieving a set of documents without ordering, whereas the latter means ordering the documents according to the relevance to the query. One of the earliest retrieval models, binary retrieval, is not able to *rank* documents but only to fetch documents according to the presence of query terms. Other than that, most retrieval models are able to produce a relevance score based on similarity measures or probabilities and thus can rank documents. In this thesis we make no distinctions between *retrieval* and *ranking* and use them interchangeably.

A ranking model can be described as estimating a relevance score given a document D and a query Q:

$$Score(Q, D)$$

One type of retrieval model estimates the score using the *similarity* between the query representation and the document representation:

$$Score(Q, D) = Similarity(Q, D)$$

Another type is based on the probability of a document being relevant to a query. A binary variable R is used to denote relevance. It has two values: 1 or 0 (in literature the notation r and $\bar{r}$ are also used). Then documents can be ranked according to such a probability:

$$Score(Q, D) = P(R = 1|Q, D) \tag{2.8}$$

This type of retrieval model is also called a probabilistic model. The theoretical foundation is justified by the Probability Ranking Principle (PRP) [168] which formally describes the retrieval and ranking task as estimating the *probability* $P(R = 1|Q, D)$. The principle lays the high-level foundation for developing retrieval models using probability theory but does not specify a method for estimating the probability.

Some models assume that query terms are independent to each other and adopt a simpler way to estimate the relevance score:

$$Score(Q, D) = \sum_{q \in Q} Score(q, D)$$

With this assumption, many term-level scores can be pre-computed and cached with the inverted index for better efficiency.

2.2.1 Boolean Retrieval

The Boolean retrieval model was one of the earliest retrieval models. It retrieves documents based on the presence of query terms in that document. All the query terms are combined using AND, OR, and NOT logic operators. Documents satisfy all the logic conditions are finally retrieved. Using "teen" and "pregnancy" as an example, the query *teen AND pregnancy* will retrieve doc 1, doc 2 and doc 4 which contain both terms. The query *teen OR pregnancy* will retrieve doc 1, doc 2, doc 4, doc 5, and doc 6 which contain either query term. The process is illustrated in Figure 2.6.

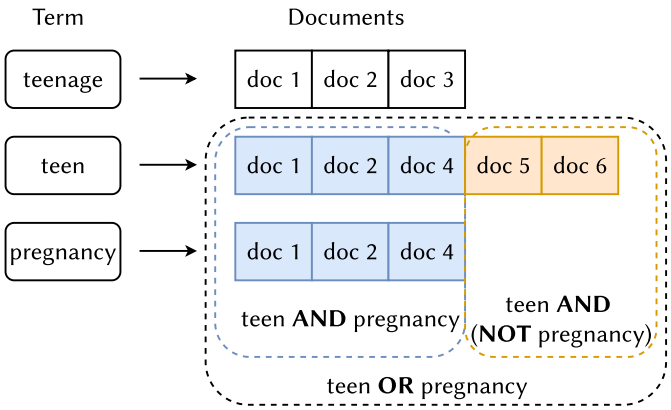


Figure 2.6: Boolean retrieval. "*teen **AND** pregnancy*" retrieves doc 1, doc 2, and doc 4. "*teen **OR** pregnancy*" retrieves doc 1, doc 2, doc 4, doc 5, and doc 6.

The Boolean model is a perfect example of leveraging the inverted index to quickly prune unmatched documents. However, the shortcomings are also clear. The use of logic operators imposes extra burden on users as they need to try multiple possibilities of combining the terms. When the number of terms become large, formulating a reasonable query becomes challenging. This is probably also why the application is limited to fields like medical search [94] where the users are usually domain experts. Another shortcoming is that it does not rank documents. When many documents satisfy the query condition, all the documents are considered equally relevant and returned to the user.

2.2.2 TF-IDF

The TF-IDF score is a numerical statistic for measuring word importance. It is widely applied in information retrieval (e.g. the vector space model and BM25 which are discussed later), keyword extraction [110], topic modeling (e.g. the latent dirichlet allocation [24]), recommender systems [11] and so forth.

The score is a combination of two components. The TF part stands for *term frequency* which describes how frequent a term appears in a document. Intuitively, if a term occurs frequently in a document, it may be an indication of high importance. The IDF part stands for *inverse document*

frequency which describes how discriminative a term is. If a term appears in too many documents, this term is not helpful in distinguishing documents. For example, "the", "a", and "an" almost appear in every document, but they are not informative. Hence, the *inverse* of document frequency is used to favor rare words.

The exact values of TF and IDF can be computed in various ways. The simplest form might be:

$$\begin{aligned} TFIDF(D,t) &= TF(D,t) \cdot IDF(t) \\ TF(D,t) &= \frac{f_{t,D}}{\sum_{t' \in D} f_{t',D}} \\ IDF(t) &= \log \frac{N}{n_t} \end{aligned} \tag{2.9}$$

where $f_{t,D}$ is the number of occurrences of term t in document D, N is the total number of documents in the collection, and n_t is the number of documents containing term t.

Many variations have been proposed based on different motivations. The raw term frequency count can be directly used: $TF(D,t) = f_{t,D}$. However, as the number of occurrences increases, the increase in importance might become marginal, so the raw frequency count can be discounted: $TF(D,t) = \log(f_{t,D} + 1)$. For IDF, to facilitate the calculation of out-of-vocabulary terms, the n_t value is often smoothed: $IDF(t) = \log \frac{N}{n_t+1}$.

Despite these slightly different forms, the property of TF-IDF has been consistent. Hereafter, we will refer to TF-IDF as a general concept unless otherwise discussed.

To rank documents, the TF-IDF score of each query term can be aggregated [125]:

$$Score_{TFIDF} = \sum_{q \in Q} TFIDF(D,q) \tag{2.10}$$

2.2.3 Vector Space Model

The vector space model (VSM) [181] converts queries and documents into a latent vector space and ranks documents based on the similarity between the query vector representation and the document vector representation. This is also a general paradigm followed in other work which use various methods to yield the representations. For example, Huang et al. [85] used a deep neural network to learn the vector representations of queries and documents and then ranked documents by their cosine similarity to the query.

In VSM, the vector space is v-dimensional where v is the size of the entire vocabulary. So, regardless of the length of a query or a document, their vector representations have the same shape as shown in Figure 2.7. The feature values q_i and d_i can have many forms:

- Binary — 1 if the term is present and 0 otherwise.
- Raw count — the number of occurrences of that the term appears.
- TF-IDF— the TF-IDF score of the term.

$$
\begin{array}{lccccc}
 & \text{Term 1} & \text{Term 2} & \cdots & \text{Term v} & \\
D = & (\ d_1, & d_2, & \cdots, & d_v &) \\
Q = & (\ q_1, & q_2, & \cdots, & q_v &)
\end{array}
$$

Figure 2.7: The vector representations of a query and a document. They both have the same number of elements v which is the number of terms in the vocabulary. When the v is large, the vectors are very sparse where most elements are zero.

Given the vector representations, documents can be ranked according to the similarity to the query:

$$
\begin{aligned}
Score_{VSM}(\mathbf{Q}, \mathbf{D}) &= \text{cosine}(\mathbf{Q}, \mathbf{D}) \\
&= \frac{\mathbf{Q} \cdot \mathbf{D}}{\|\mathbf{Q}\| \cdot \|\mathbf{D}\|} \\
&= \frac{\sum_i^v q_i \cdot d_i}{\sqrt{\sum_i^v q_i^2} \cdot \sqrt{\sum_i^v d_i^2}}
\end{aligned}
\tag{2.11}
$$

2.2.4 BM25

Robertson et al. [173] proposed BM25 which empirically combines *term frequency* and *inverse document frequency*. In the original paper, the scoring function is defined as:

$$
\begin{aligned}
Score_{BM25}(Q, D) &= \sum_{t \in Q \cap D} \frac{TF_{BM25}(D, t)}{K + TF_{BM25}(D, t)} \cdot IDF_{BM25}(t) \\
K &= k_1(1 - B + B \cdot \frac{l_D}{l_{avg}}) \\
TF_{BM25}(D, t) &= \frac{f_{t,D}}{\sum_{t' \in D} f_{t',D}} \\
IDF_{BM25}(t) &= \log(\frac{N - n_t + 0.5}{n_t + 0.5})
\end{aligned}
\tag{2.12}
$$

where $f_{D,t}$ is the frequency of occurrences of t within D, l_D is the length of document D, l_{avg} is the average document length of the entire collection, N is the total number of documents, and n_t is the number of documents containing t. k_1 and B are two hyperparameters which are collection specific.

Since BM25 is effective while still being efficient, it is now often used to retrieve an initial set of candidate documents for more expensive neural ranking models in later stages. The field-based variation BM25F [221] is also popular for collections with field information.

2.2.5 Query Likelihood

Query likelihood [160] is a probabilistic ranking model based on language models. A *language model* describes the probability a word w is generated. In a document about "information retrieval", the words "information" and "retrieval" are likely to appear very frequently, so they are expected to have high probabilities of being used. The probability distribution of the entire vocabulary is called the language model. In fact, a language model describes the word preference of a piece of text. When a user writes a query, they may have a set of candidate words for the hidden information need. When an author writes an article, they also have a vocabulary for the topic of the article. So, a language model can be induced from various text source – a query, a document, or a collection.

The simplest way of inducing a language model is to use word frequencies, i.e. frequent words will have higher probabilities than less frequent ones. Given a document D, the document language model is defined as:

$$\begin{aligned} P(w|D) &= \frac{f_{w,D}}{\sum_{w' \in D} f_{w',D}} \\ &= \frac{f_{w,D}}{|D|} \end{aligned} \tag{2.13}$$

With a document language model, we are able to estimate the probability of a query Q being generated from the language model assuming that query terms are independent of each other:

$$P(Q|D) = \prod_{q \in Q} P(q|D)$$

This probability is the *query likelihood* and can be used to rank documents. In practice, a rank-equivalent form is used in order to improve the numerical stability:

$$\begin{aligned} Score_{QL}(Q, D) &= \log P(Q|D) \\ &= \sum_{q \in Q} \log P(q|D) \end{aligned} \tag{2.14}$$

A document language model estimated from Eq 2.13 is subject to the sparsity of the document. Compared to the vocabulary, the number of unique terms in a document is rather small, which means most words have a probability of zero. Intuitively, this language model induced solely from the document may not be an accurate language model. Although other words happen to have not occurred in the document, they should have a small probability but not zero. To alleviate the problem, a collection language model is often used to smooth the probability. Given a collection of documents C, the collection language model $P(w|C)$ is induced by counting the term frequencies in the entire collection, so it provides a default probability for every word in the collection. The smoothed language model is modified as:

$$P'(w|D) = (1 - \alpha)P(w|D) + \alpha P(w|C) \tag{2.15}$$

where α is a smoothing coefficient which can be a constant or document dependent. When α is constant, the smoothing is known as *Jelinek-Mercer* smoothing. According to [224], *Jelinek-Mercer* smoothing tends to perform better for long queries. The well-known *Dirichlet* smoothing defines $\alpha = \frac{\mu}{|D|+\mu}$ with μ being a hyperparameter. The contribution of $P(w|D)$ is then proportional to the document length $|D|$. The longer the document is, the more data it provides to estimate the language model and hence the more accurate the estimation is. It is observed that *Dirichlet* smoothing performs better for short queries [224]. The final $Dirichlet$-smoothed document language model is:

$$\begin{aligned} P'(w|D) &= (1-\alpha|D)P(w|D) + \alpha P(w|C) \\ &= (1 - \frac{\mu}{|D|+\mu})\frac{f_{w,D}}{|D|} + \frac{\mu}{|D|+\mu}\frac{f_{w,C}}{|C|} \\ &= \frac{f_{w,D} + \mu\frac{f_{w,C}}{|C|}}{|D|+\mu} \end{aligned} \tag{2.16}$$

Substituting this definition for Eq 2.14 results in the final equation of query likelihood ranking.

Theoretical Formulation. The lack of an explicit relevance variable in deriving the query likelihood model has been questioned by researchers at its emergence. From a probability theory perspective, we would like to use $P(R = 1|Q, D)$ (recall Eq 2.8 which is based on the Probability Ranking Principle) to rank documents, but it is unclear how this probability is connected to Eq 2.14. Lafferty and Zhai [105] completed this theoretical connection and derived Eq 2.14 in a probabilistic framework. Applying the Bayes' rule multiple times, the probability of relevance $P(R|D, Q)$ is factored as follows:

$$\begin{aligned} Score_{QL}(Q, D) &= P(R = 1|D, Q) \\ &\propto \log \frac{P(R = 1|D, Q)}{P(R = 0|D, Q)} \\ &= \log \frac{P(Q|D, R = 1)}{P(D|Q, R = 0)} + \log \frac{P(R = 1|D)}{P(R = 0|D)} \end{aligned} \tag{2.17}$$

Now we further factor the two components respectively.

Assuming D and Q are independent on event $R = 0$, in other words, assuming D and Q are independent if they are non-relevant:

$$\begin{aligned} \log \frac{P(Q|D, R = 1)}{P(D|Q, R = 0)} &= \log \frac{P(Q|D, R = 1)}{P(Q|R = 0)} \\ &\propto \log P(Q|D, R = 1) \end{aligned}$$

Additionally, assuming D and R are independent, the last component becomes a constant and can be ignored:

$$\log \frac{P(R=1|D)}{P(R=0|D)} = \log \frac{P(R=1)}{P(R=0)}$$

Substituting these items for the components in Eq 2.17, we get

$$Score_{QL}(Q,D) \propto \log P(Q|D,R=1) \tag{2.18}$$

which links back to Eq 2.14 as introduced in Lafferty and Zhai [105].

Language modeling is a core foundation of this thesis. The language models we discussed here can be referred to as *statistical language models*, in order to distinguish them from the more recent *neural language models*. *Statistical language modeling* is the seminal work of Chapter 3 and *neural language modeling* is a key component of Chapter 5. We will discuss neural language models in Sec 2.4.3 where we introduce Transformers.

2.2.6 Sequential Dependence Model

Word dependencies are crucial for understanding a piece of text. Using "information retrieval" as an example, the probability that "retrieval" occurs after "information" is probably higher than when it appears without a context. However, many retrieval models disregard this information for simplicity – words are assumed to be independent to each other. To capture this key information, Metzler and Croft [127] proposed a general framework which incorporates term dependencies. This framework can be considered as a generalization of three language models: *unigram* language models [160], *bigram* language models [190], and finally *biterm* language models [192].

Table 2.2: Language models for an ordered sequence of m words $w_1w_2w_3...w_m$.

Type	Definition
Unigram	$P_{ug}(w_1w_2w_3...w_m\|D) = P(w_1\|D)P(w_2\|D)P(w_3\|D)...P(w_m\|D)$
Bigram	$P_{bg}(w_1w_2w_3...w_m\|D) = P(w_1\|D)P(w_2\|w_1|D)P(w_3\|w_2,D)...P(w_m\|w_{m-1},D)$
Trigram	$P_{tg}(w_1w_2w_3...w_m\|D) = P(w_1\|D)P(w_2\|w_1|D)P(w_3\|w_1w_2,D)...P(w_m\|w_{m-2}w_{m-1},D)$
n-gram	$P_{ng}(w_1w_2w_3...w_m\|D) = \prod_i^m P(w_i\|w_{i-(n-1)}...w_{i-1},D)$

The *unigram* and *bigram* language models are two instances of the general n-gram language model which conditions a word on *n-1* previous words. We summarize their definitions in Table 2.2. The estimation of $P(w_i|w_{i-(n-1)}...w_{i-1},D)$ is estimated from n-gram frequency count and is a generalization of Eq 2.13:

$$P(w_i|w_{i-(n-1)}...w_{i-1},D) = \frac{f_{w_{i-(n-1)}...w_{i-1}w_i,D}}{f_{w_{i-(n-1)}...w_{i-1},D}}$$

Note that smoothing techniques can be incorporated into the estimation same as Eq 2.15. We omit the smoothing part for simplicity.

The *biterm* language model is slightly different to a *bigram* language model in that it relaxes the term ordering constraint. So the *biterm* probability can be estimated as from the *bigram* probability of the swapped words :

$$P_{bt}(w_{i-1}w_i|D) = \frac{1}{2}P_{bg}(w_{i-1}w_i|D) + \frac{1}{2}P_{bg}(w_i w_{i-1}|D)$$

The sequential dependence model (SDM) now can be formally defined:

$$\begin{aligned} Score_{SDM}(Q,D) = & \sum_{q_i \in Q} \lambda_U \log P_{ug}(q_i|D) + \\ & \sum_{q_i q_{i+1} \in Q} \lambda_G \log P_{bg}(q_i q_{i+1}|D) + \\ & \sum_{q_i q_{i+1} \in Q} \lambda_T \log P_{bt}(q_i q_{i+1}|D) \end{aligned} \tag{2.19}$$

where λ_U, λ_G and λ_T are three hyperparameters.

2.3 Neural Ranking Models

2.3.1 Neural Networks for Ranking

Neural networks have achieved breakthrough progress in computer vision (e.g. Krizhevsky et al. [103] for image recognition) and natural language processing (e.g. Bahdanau et al. [6] for machine translation) in the past few years. We have also witnessed exciting improvements in IR by applying neural networks to ranking. In 2019, TREC Deep Learning Track has seen that the best neural network model outperforms the best traditional model by a huge margin of 37.4% [44]. In 2020, the gap was increased to 42% [45]. The significant progress can not be contributed by any single technique. From the literature, we can see that the progress in ranking has gone through three stages, each of which has contributed a key technique to get us to where we are today:

- ✦ Learning-to-rank
- ✦ Deep neural ranking models
- ✦ Transformer ranking models

The first stage is learning-to-rank which applies machine learning in ranking. At this stage, the commonly used loss paradigms — *pointwise*, *pairwise* and *listwise* — were developed. Nowadays, these loss functions still play a fundamental role in training neural ranking models. Some early learning-to-rank work has also explored using neural networks [27]. A property of techniques from this stage is the use of manually designed features. Gallagher et al. [71] released a

feature extraction framework *fxt* which has 448 available features. However, the feature-based models is also limited by feature engineering.

The second stage is applying deep neural networks. This stage has a focus on the design of network architectures. Although deep neural ranking models can be considered as a learning-to-rank technique in a broad sense, one major advantage that separates neural ranking models from classic learning-to-rank models is the ability to learn from raw text without having to design features manually. In the beginning of the stage, the focus was mainly on representation learning than the design of ranking functions. Queries and documents are projected into a latent space individually and the relevance was measured using similarity metrics, which was similar to the vector space model (Sec 2.2.3). Later, neural networks were incorporated in an end-to-end ranking architecture where the ranking function is also implemented using neural networks. Guo et al. [76], Mitra and Craswell [131] comprehensively summarized work at this time period. The application of deep neural ranking models was an attempt to transfer the success in NLP to IR. However, more noticeable improvements were not observed until recently.

The third stage was marked by the emergence of the Transformer [201] and neural language models (more specifically BERT [57]). Transformers are a breakthrough in the design of network architectures for language related tasks, using an *attention* mechanism [6]. Transformers have remarkably improved the efficiency of the attention mechanism through self-attention variants which we will introduce in detail in Sec 2.4.1. Later, Devlin et al. [57] used language modeling tasks to initialize the transformer parameters with generalized language knowledge. Their pre-trained model turned out to be easily transferable to many text related tasks including ranking. Nogueira and Cho [145] was perhaps the first to successfully use BERT for ranking. Since their work, Transformers-based ranking models have dominated in IR.

In the rest of this subsection, we provide a high-level overview of neural ranking architectures and elementary learning techniques.

2.3.2 Neural Ranking Framework

A neural ranking model M produces a relevance score given a query Q and a document D. The framework is illustrated in Figure 2.8 and can be formally described using the following formula as abstracted by Guo et al. [76]:

$$\begin{aligned} Score_{neural}(Q, D) &= M(Q, D) \\ &= g(\psi(Q), \phi(D), \eta(Q, D)) \end{aligned} \tag{2.20}$$

where g computes the score based on input features, ψ, ϕ and η are representation learning functions for the query, document, query-document pair respectively.

More specifically, for learning-to-rank models, ψ and ϕ extract *static* features such as number of terms and average term length; η is an interaction function which extract features such as the BM25 scores of the query-document pair. We list several example features in Table 2.3.

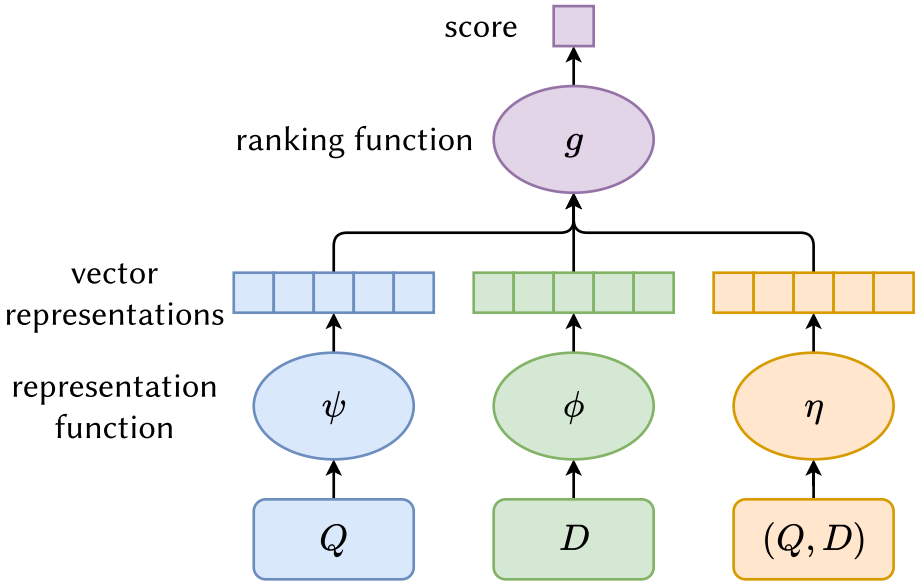


Figure 2.8: A highly abstractive generalized framework of neural ranking models.

For neural ranking models, according to Guo et al. [76], there are two types of models based on the underlying assumptions on relevance.

- *Representation*-focused models are illustrated in Figure 2.9a. They have the assumption that relevance is determined by the inherent meaning of the text. These models learn the representations of queries and documents separately and use simple metrics to measure the similarity of the representations. The representation functions ψ and ϕ are neural networks which project text into a latent space while η is not used.
- *Interaction*-focused models are illustrated in Figure 2.9b. They assume that relevance is more about the *relation* between queries and documents. So, queries and documents should not be processed separately. There are many ways of capturing the interaction between a query and a document. For example, they can be concatenated together and fed into a ranking model as one sequence. In this case, ψ and ϕ are not used while η yields a joint representation.

Many details are also hidden in the design of the networks ψ, ϕ, η and g. A few popular choices are feed-forward networks [183], convolutional networks [68], recurrent networks [81, 176], and Transformers.

2.3.3 Loss Functions

A loss function is a crucial part of a neural ranking model. It measures how well a model's predictions reflect ground truth labels and provides directions for optimizing the model. According

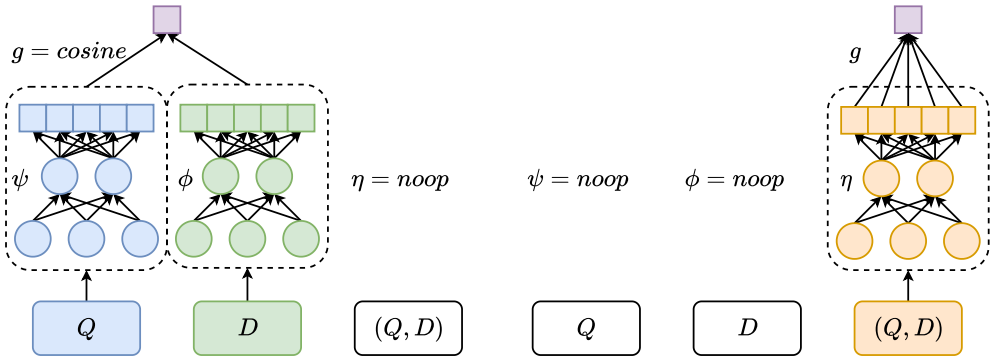
Table 2.3: Examples of neural ranking features. IDF is defined in Eq 2.9; $Score_{TFIDF}$ is defined in Eq 2.10; $Score_{BM25}$ is defined in Eq 2.12; $Score_{QL}$ is defined in Eq 2.14. $|X|$ denotes the number of terms contained in X; $PageRank$ is a measure of importance of web pages [154]; $f_{q,D}$ is the number of occurrences of q in D.

	Name	Definition
Query feature	Query length	$\psi(Q) = \|Q\|$
	Mean IDF	$\psi(Q) = \frac{1}{\|Q\|} \sum_{q \in Q} IDF(q)$
Document feature	Document length	$\phi(D) = \|D\|$
	Page rank	$\phi(D) = PageRank(D)$
Query dependent feature	TF	$\eta(Q, D) = \sum_{q \in Q} f_{q,D}$
	TF-IDF	$\eta(Q, D) = Score_{TFIDF}(Q, D)$
	BM25	$\eta(Q, D) = Score_{BM25}(Q, D)$
	Query likelihood score	$\eta(Q, D) = Score_{QL}(Q, D)$

to Liu [119], there are three types of loss functions that are most commonly used for training a ranking model: *pointwise*, *pairwise*, and *listwise* losses, which use a single, a pair and a list of document scores to calculate the loss respectively. It is worth noting that the terminology does not refer to network architectures. The only requirement on a network is to produce a real-valued score given a query and a document. Loss functions discussed here are orthogonal to and can be applied to any network architectures described by Eq 2.20. For pointwise losses, we apply the network on one query-document pair to get *one* score and calculate a loss between the score and the ground truth; for pairwise losses, we apply the network twice on two query-document pairs to get *two* scores and calculate a loss based on the rank determined by the scores; for listwise losses, we apply the network multiple times to get a *list* of scores and calculate a loss based on the list of scores. In the literature, the term "pairwise" might be used in another way. For example, Nogueira et al. [147] proposed a "pairwise" ranking model which takes one query and a pair of documents and predicts if the document order is correct.

We use similar notation to the machine learning literature to describe loss functions. The model M takes as input x which can be a feature vector, as in learning-to-rank or a piece of raw text, as in more recent deep neural ranking models. The relevance judgment for x is denoted as y, which can be binary (0 for non-relevant and 1 for relevant) or graded (for example, Clarke et al. [36] used 0 for "not relevant", 1 for "not relevant, but reasonable", 2 for "relevant", and 3 for "highly relevant"). The only constraint on the model is that the output should be a real value which we denote as $\hat{y}$. For simplicity, we also assume that $\hat{y}$ ranges from 0 to 1:

$$\hat{y} = M(Q, D) \in [0, 1]$$



(a) Neural ranking with a focus on representation learning.

(b) Neural ranking with a focus on query-document interactions.

Figure 2.9: Instance of the neural ranking models with different focuses. The left panel has a focus on representation learning. The right panel has a focus on the query-document interaction and the ranking function.

although some losses do not have this constraint. One approach to convert an unbounded real value v is applying a sigmoid function σ:

$$\sigma(v) = \frac{1}{1 + e^{-v}} \tag{2.21}$$

We express specific loss functions using the notation of $\hat{y}$ and y. To simplify the notation, we only define a loss using a single training instance. Our definitions can be easily extended to batched training by averaging or weighted-averaging the losses in the batch.

Pointwise Loss. Pointwise losses are a direct application of regression and binary classification in IR. If we view ranking as a regression problem where the model outputs a real-valued relevance score, then the loss can be a mean squared error (MSE) loss:

$$L_{MSE}(y, \hat{y}) = (\hat{y} - y)^2 \tag{2.22}$$

Ranking can also be viewed as a binary classification problem where the model is predicting whether a document belongs to the relevant class or non-relevant class. This formulation fits perfectly for training data with binary judgments while graded judgments need to be mapped to binary according to some rules. Most commonly, non-zero judgments are mapped to 1. The loss is a binary cross entropy (BCE) loss:

$$L_{BCE}(y, \hat{y}) = -(y \log \hat{y} + (1 - y) \log P(1 - \hat{y})) \tag{2.23}$$

Pointwise losses are a natural extension of classic machine learning approaches which were devised decades ago. It may suffer from imbalanced data, as commonly observed in machine learning. Or even worse, in an IR setting, most documents belong to the non-relevant class. In such a case, down-sampling the non-relevant class is often required. In addition, the intrinsic problem of ranking is the *order* of documents instead of the classification or the scores. Is a document with a relevance score of 4 two times more relevant than a score of 2? It is hard to say as *relevance* is subjective. With pointwise approaches, ordering can not be easily incorporated during training which often leads to sub-optimal models. Moreover, the BCE loss (Eq 2.23) needs a binary label and this is often done by mapping non-zero graded judgments to 1 which has completely discarded the orders within those judgments. To address the problem, pairwise and listwise losses were proposed to directly model *orders* for ranking.

Pairwise Loss. Pairwise loss has an emphasis on the rank of a pair of documents instead of their exact labels. Given two inputs x_1 and x_2 with the labels $y_1 > y_2$, the objective is also predicting the relevance score so that $\hat{y}_1 > \hat{y}_2$. One of the most popular pairwise losses is hinge loss:

$$L_{Hinge}(\hat{y}_1, \hat{y}_2) = max(0, 1 - (\hat{y}_1 - \hat{y}_2)) \tag{2.24}$$

Note that the original labels y_1 and y_2 are not used, and only the order of the predictions matters.

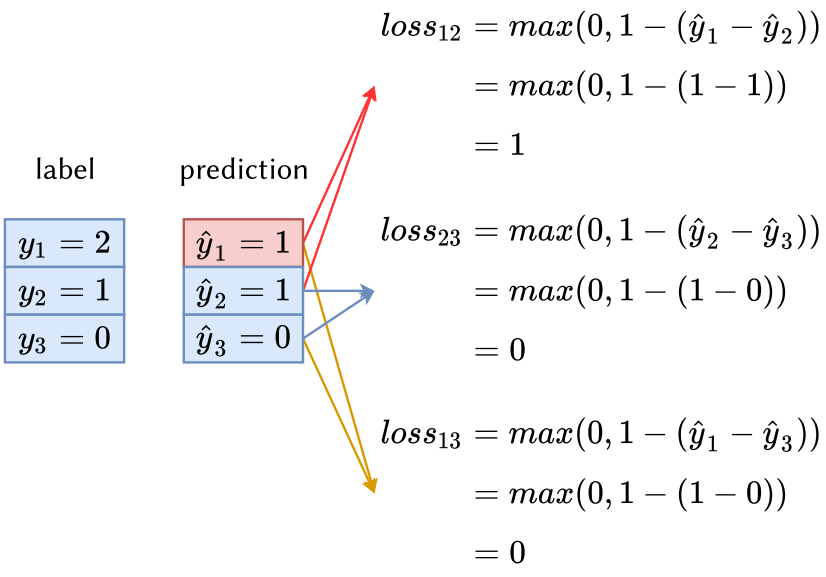


Figure 2.10: An example of hinge loss.

Another popular pairwise loss is the RankNet loss [26], shown in Eq 2.25. Since it uses a logistic function to map model outputs to probabilities, it is also referred to as the pairwise logistic

loss.

$$L_{RankNet}(\hat{y}_1, \hat{y}_2) = -\log P(\hat{y}_1 \triangleright \hat{y}_2)$$
$$P(\hat{y}_1 \triangleright \hat{y}_2) = \frac{1}{1 + exp(\hat{y}_i - \hat{y}_2)} \tag{2.25}$$

where $\hat{y}_1 \triangleright \hat{y}_2$ denotes that $\hat{y}_1$ is ranked higher than $\hat{y}_2$.

Although pairwise loss takes only a pair of instances to calculate the loss, it can be easily seen that it works well for a list of instances as well, if we turn a list into multiple pairs. Figure 2.10 shows an example of how hinge loss work in a listwise setting. For the three instances, we can construct three pairs. For the pair $(\hat{y}_2, \hat{y}_3)$ and $(\hat{y}_1, \hat{y}_3)$, the current scores can order them correctly, so the corresponding losses $loss_{23}$ and $loss_{13}$ are 0. The only problematic ordering is between $\hat{y}_1$ and $\hat{y}_2$ given the current scores. With $loss_{12} = 1$, the model is tuned to assign a higher score to $\hat{y}_1$ to minimize the overall loss and that will lead to the correct ordering of the entire list.

The pairwise approach may also be subject to imbalanced data as was the pointwise approach. While relevant-non-relevant ratio is always $1 : 1$ in a pairwise setting, the imbalance can still be caused by relevance judgments at the query level and be amplified due to the construction of pairs. A query with more relevance judgments tends to have significantly more pairs [162]. For example, if a query has 5 relevant documents at different levels, we can construct 10 ($5 \times 4/2 = 10$) valid pairs. If a different query has 20 judgments at different levels, the number of pairs becomes 190 ($29 \times 19/2 = 190$) which is 18 times more than the previous query. In such a case, the training is dominated by queries with a large number of relevant documents. To alleviate the query-level imbalance, query-level normalization needs to be considered, where the loss contribution from a query is divided by the number of pairs for that query.

Pairwise losses consider every pair equally important. This is not true for many IR metrics which have more penalties at top positions such as NDCG and MAP. Ideally, mixing up position 1 with position 100 should result in a larger loss compared to mixing up position 50 with position 100, but it is not taken into consideration in a pairwise setting. To tackle these issues, researchers have studied listwise losses which can use equal amount of documents of a query and are sensitive to rank position.

Listwise Loss. A listwise loss also has an emphasis on the rank of documents instead of their exact labels. But listwise losses are able to rank a list of documents instead of just a pair. Ideally, both pairwise and listwise losses produce the optimal ranking when minimized. The listwise approach directly incorporate the rank of a list of documents whereas the pairwise approach indirectly. In a listwise setting, a list of inputs $X = \{x_1, x_2, \dots\}$ are sorted according to the labels $Y = \{y_1, y_2, \dots\}$ where $y_1 \geq y_2 \geq \dots$.

The ListMLE [213] loss aims at maximizing the probability of selecting each document from the ranked list $Y = \{y_1, y_2, \dots\}$. Consider the following process: for $\{y_1, y_2, \dots\}$, where y_1 has the highest probability of being selected; for the rest $\{y_2, \dots\}$, y_2 has the highest probability of being selected; this process continues until every document is selected. Xia et al. [213] show that by maximizing the product of every probability, the optimal ranking is achieved. Equivalently,

the loss is defined to minimize the negative sum of the probabilities:

$$L_{ListMLE}(\hat{Y}, Y) = -\sum_i \log P(\hat{y}_i)$$
$$P(\hat{y}_i) = \frac{exp(\hat{y}_i)}{\sum_{j=i} exp(\hat{y}_j)} \tag{2.26}$$

The key to this loss is the estimation of the probability of selecting the i_{th} document $P(\hat{y}_i)$. Figure 2.11 shows two examples of the calculation which differ at the prediction of $\hat{y}_1$. With $\hat{y}_1 = 1$, the overall probability is 0.309. With $\hat{y}_1 = 2$, the overall probability is 0.486. When the loss is minimized, the probability is maximized, and the model is able to produce an optimal rank.

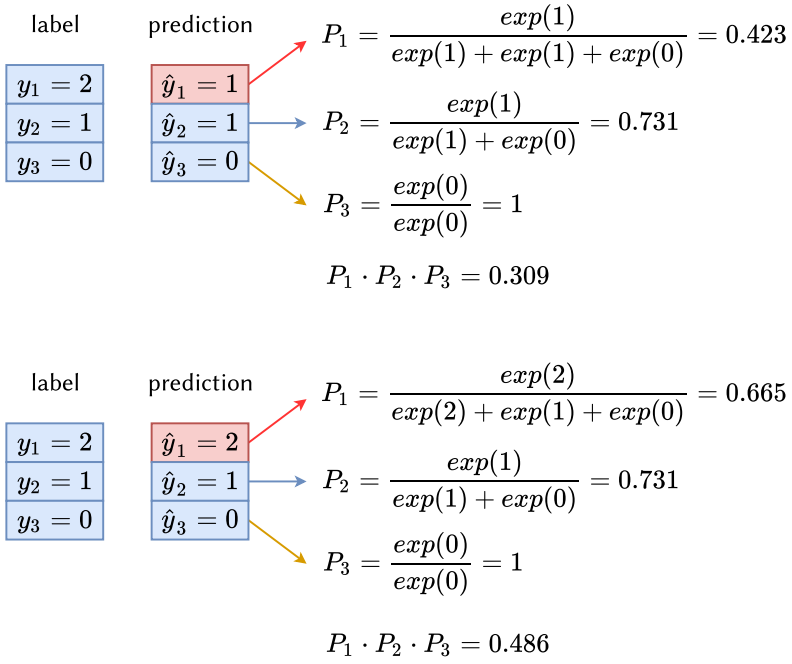


Figure 2.11: An example of ListMLE loss. The bottom prediction is superior to the top prediction as documents can be ranked correctly.

Cao et al. [30] proposed ListNet which measures the cross entropy between two top-one probability distributions estimated from the ground truth and model predictions. The top-one

probability represents the probability of a document being ranked at the topmost position.

$$
\begin{aligned}
L_{ListNet}(\hat{Y}, Y) &= -\sum_i P(y_i) \log P(\hat{y}_i) \\
P(y_i) &= \frac{y_i}{\sum_j y_j} \\
P(\hat{y}_i) &= \frac{exp(\hat{y}_i)}{\sum_j exp(\hat{y}_j)}
\end{aligned}
\tag{2.27}
$$

$P(y_i)$ denotes the ideal top-one probability of the i_{th} document given the ground truth; $P(\hat{y}_i)$ denotes the top-one probability given the model prediction. Figure 2.12 shows how relevance labels and predictions are mapped to probabilities. The loss is not at its minimum until $\hat{y}_1 = 2$ which produce the correct rank.

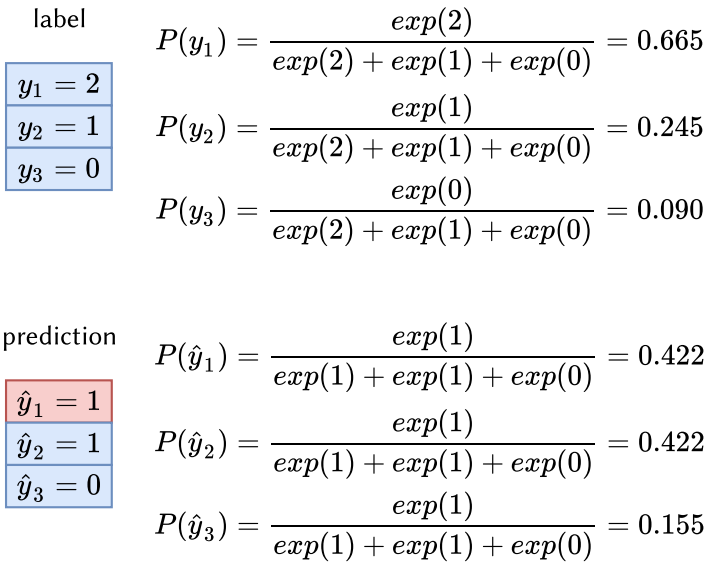


Figure 2.12: An example of ListNet loss. Labels and predictions are mapped to two probability distributions of preferences. The loss is minimized when the bottom probability distribution is identical to the top distribution.

Listwise losses are considered more effective than pairwise and pointwise approaches which discards ordering information or models ordering indirectly. But in a binary relevance setting, for example MS-MARCO passage ranking [143], a lot of ties exist. Listwise approaches may not be superior to pairwise approaches due to the lack of informative orders. Additionally, the effectiveness comes at the cost of efficiency. Listwise losses have higher computational complexities and are more suitable to be placed at later stages of a multi-stage ranking pipeline where the number of documents to rank is considerably small.

2.3.4 Network Optimization

A neural network is usually optimized using batched stochastic gradient descent, which is an optimization algorithm that minimizes a differentiable function. Although many machine learning techniques use gradient descent as a black box tool, understanding the dynamics sheds some light on tuning hyper-parameters such as learning rate when designing a neural ranking model. We refer readers to Ruder [175] for a comprehensive introduction to the technique and provide an intuitive description here.

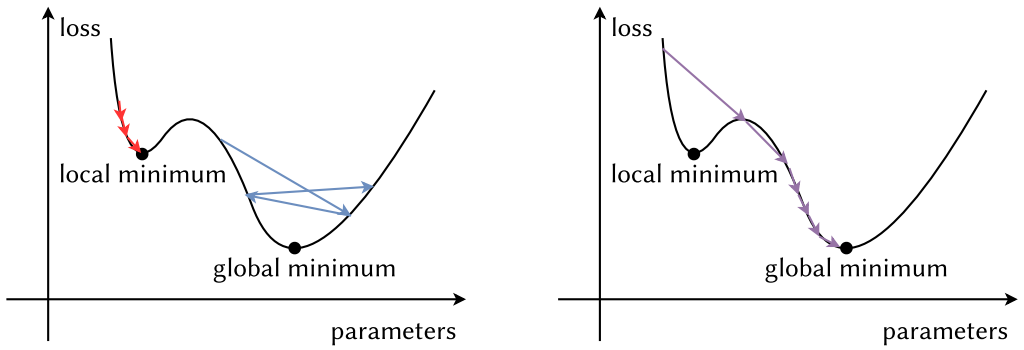
Given a loss function, or objective function which we do not distinguish in this thesis, $L(\theta)$ parameterized by θ, the objective is to find an optimal set of parameters θ^* which minimize L:

$$\theta^* = \arg\min_{\theta} L(\theta) \tag{2.28}$$

Gradient descent works by iteratively updating the parameters θ along the opposite direction of the gradient of the function which leads to a minimum through the steepest and fastest route. The update can be described by Eq 2.29.

$$\theta = \theta - \lambda \nabla_{\theta} L(\theta) \tag{2.29}$$

The adjustment is composed of two components: the direction $\nabla_{\theta} L(\theta)$ and the step size λ. $\nabla_{\theta} L(\theta)$ denotes the gradient of $L(\theta)$ with respect to θ and dictates the steepest direction of descent in the parameter space. λ is the *learning rate* and dictates how much we move along that direction. Gradient descent is analogous to a hypothetical scenario where we try to descend a mountain as fast as possible. At any location, we need to decide the direction to move and how much to move along that direction. As we move, the direction may no longer be optimal, and we need to adjust again.



(a) Too small or too large learning rates lead to sub-optimal solutions.

(b) Adaptive learning rate decays gradually to reach the global minimum.

Figure 2.13: Examples of gradient descent with different strategies for learning rate.

Figure 2.13 illustrates the impact of learning rates in minimizing the loss parameterized by one parameter. In fact, modern neural ranking models often contain millions or even billions of parameters which make it extremely hard to reach the global minimum. Figure 2.13a shows that, if the learning rate is too small (red color) or too large (blue color), the optimization may be stuck at some local minimum or not converge as it approaches the global minimum. Ideally, a proper learning rate should sometimes be large enough to escape the local minimum and be small enough as the model approaches the global minimum, as shown in Figure 2.13b. Many strategies have been proposed to tackle the problem. For example, popular approaches like Adam [100] and AdamW [122] compute an adaptive learning rate for each parameter. The descending process is like a ball rolling down a hill. If the hill is steep, the ball accelerates and the learning rate becomes larger; as the hill gets more flat close to the bottom, the momentum is reduced due to the friction with the ground and the learning rate becomes smaller.

The training dynamics also shed some light on finetuning pretrained models which we will discuss shortly. Transformers are often heavily pretrained for various language modeling tasks over millions of steps. With a pretrained model, a large learning rate may not work well as it drives the parameters away from its current minimum and lead to sub-optimal solution. We have observed in our work that the proper learning rates for finetuning are usually orders of magnitude smaller than the pretraining stage.

2.4 Sequence Models

Text is one kind of sequence data in the real world. Many others such as music, videos, or DNA sequences are common too. According to whether input and output is sequence data, sequence tasks can be classified as one-to-many, many-to-one, and many-to-many. One-to-one is the regular feed-forward neural networks formulation and can be considered a special case of sequence tasks but is not the focus of this section. Table 2.4 shows several examples of sequence tasks. Story generation can be formulated as a one-to-many task. The output of story generation is a piece of text and the input might be a genre identifier. Image captioning takes an image and outputs a caption. Sentiment analysis is a typical many-to-one task where the output is scores or classifications indicating positive or negative emotion. Queries and documents are also sequence data. Ranking is essentially estimating the relevance between two sequences of text. Many-to-many tasks are more challenging as the input end and the output end are both sequences. Named entity recognition assigns a label to every term in the sequence. Machine translation, summarization, and query generation map source text to target text depending on the task formulation.

Sequence data has two challenging properties that are nontrivial to model: variable lengths and long-term dependencies. A news article may contain thousands of words while the concluding text could depend on only the early introduction text. These properties of sequence data have motivated a special kind of models: *sequence models*. In this section, we review various sequence models used in IR. Particularly, our work heavily relies on the transformer architecture [201]. Strictly speaking, transformer-based ranking models are a type of neural ranking models, but we have a dedicated section to introduce them instead of discussing them in earlier sections due to

Table 2.4: Examples of sequence modeling for text.

Type	Task	Input	Output
One-to-many	story generation	genre	text
One-to-many	image captioning	image	text
Many-to-one	sentiment analysis	sentence	classification
Many-to-one	**ranking**	query and document	score
Many-to-many	named entity recognition	text	text labels
Many-to-many	machine translation	source language	target language
Many-to-many	summarization	long paragraph	short summary
Many-to-many	**query generation**	document	query

the importance to this thesis. We will start with a brief introduction to early sequence models which preceded transformers and whose shortcomings inspired transformers. We also introduce a special type of sequence model: seq2seq whose input and output are both sequence data. In each part, we first introduce a high-level abstraction of the type of models, followed by specific examples.

2.4.1 Sequence Modeling

Sequence Representations. As a standard practice, every word in a sequence of text is represented as a *word embedding* which is a vector of a predefined dimension d, for example 512, 768, or 1024. Imagine a 3-dimensional space, a word is like a point *embedded* in such a space. The choice of the dimension is empirical and often depends on the available computational resources. Usually, the larger, the more expressive the representation is, but the computational complexity grows as the size grows. The sentence "She is eating a green apple" contains 6 words and is likely to be converted into 6 vectors of size d.

Word embeddings are often learned from unlabeled text. For example, *Word2vec* [130] proposed learning word embeddings through predicts the context words given a target word. Using "She has apple juice every day" as an example. Suppose we use 2 words on the left and right as context. Given "apple", the model is trained to predict "she", "has", "juice", and "every". When the model is trained with a different sentence "She has orange juice every day", it will also learn to predict the same context given "orange". Formally, let θ_t, θ_c represents the embeddings of the target word t and context word c. The simplest objective is to maximize the probability of predicting t given c in the vocabulary V:

$$P(c|t) = \frac{e^{\theta_c^\top \theta_t}}{\sum_{x \in V} e^{\theta_x^\top \theta_t}}$$

Eventually, "apple" and "orange" will have very similar word embeddings in order to maximize the probability.

Word2vec is an example of how vector representations of words can be learned. There are many other techniques such as the neural language modeling approach [16], GloVe [158], and more recently contextualized embeddings such as ELMo [159] and BERT [57]. Importantly, contextualized embeddings address a limitation of static embeddings by taking into consideration the *context*. Note that word2vec also uses context words to learn the embeddings, but the learned embeddings are static after training. Using the sentence "she has apple juice every day" as an example. The word "apple" on its own can refer to the fruit or a company. With static word embeddings we can not distinguish the meaning. However, the contextualized approaches yield the embeddings of "apple" conditioned on its context – "juice" is a strong indication that "apple" is the fruit. So, it is likely the contextualized embeddings of "apple" more accurately represent its real meaning. All of them can map words to real-valued vectors that contain the meaning of the words so that similar words are closer to each other in the vector space. Word embeddings are fundamental to sequence models, so, in this thesis we will use word embeddings to represent a sequence.

Recurrent Neural Networks. Recurrent neural networks (RNN) [177] are one of the earliest sequence models. An RNN processes the elements of a sequence one by one, so it can model variable-length sequences. When processing each element, an RNN maintains a *hidden state* that encodes the information of all previous elements, so it can model long-term dependencies. Figure 2.14 shows a vanilla RNN. Let $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_{t_x}}\}$ denote the embeddings of the input sequence of length t_x, $Y = \{y_1, y_2, \ldots, y_{t_y}\}$ denote the ground truth sequence in a supervised setting, $\hat{Y} = \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{t_y}\}$ denote the output sequence of length t_y. Note that t_y may or may not be equal to t_x. For example, in named entity recognition, the network produces one label for each term, so t_x is equal to t_y as illustrated in Figure 2.14. In translation, the translated text may not have the same number of terms as the original text, so t_x is not equal to t_y, which will be discussed shortly.

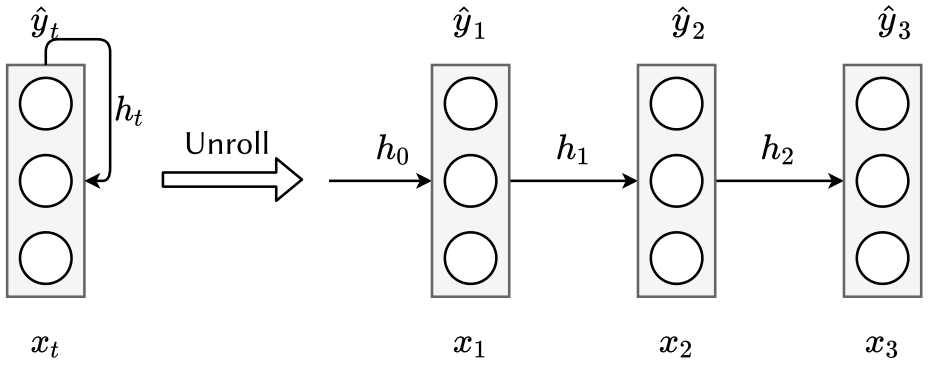


Figure 2.14: A sequence model structure shared by RNN, LSTM, and GRU.

The left panel of Figure 2.14 shows a physical computing unit called a *cell*; the right panel shows the same cell at different time steps. A cell takes as input $\mathbf{x_t}$ and the state from the previous step $\mathbf{h_{t-1}}$ and yields $\hat{y}_t$ and the new hidden state $\mathbf{h_t}$. The first hidden state $\mathbf{h_0}$ is often initialized with 0 since there is no previous step. An RNN cell is parameterized by three matrices $\mathbf{W_{hx}}$, $\mathbf{W_{ha}}$, $\mathbf{W_{hy}}$, and two bias vectors $\mathbf{b_h}$, and $\mathbf{b_o}$. A cell is described as follows:

$$\begin{aligned} \mathbf{h_t} &= \sigma(\mathbf{W_{hx}x_t} + \mathbf{W_{ha}h_{t-1}} + \mathbf{b_h}) \\ \hat{y}_t &= \sigma(\mathbf{W_{hy}h_{t-1}} + \mathbf{b_o}) \end{aligned} \tag{2.30}$$

where σ is an activation function. This high-level illustration is also shared by other sequence models such as an LSTM [81] and a GRU [34] but they differ in cell definitions.

This abstraction of sequence data has two intrinsic shortcomings. First, the hidden state h_t needs to encode all the information before the current time step t. As the sequence gets longer, the burden on h_t increases. Second, the calculations can not be parallelized due to the recurrent dependencies – to calculate h_t, we need to calculate h_{t-1} first. The first shortcoming inspired the invention of the attention mechanism and the second shortcoming later inspired the Transformer architecture.

Convolutional Neural Networks. Sequence data can also be modeled using convolutional neural networks (CNN) [68]. Essentially a convolution operation is a window (a kernel) sliding through the sequence and taking dot products of local regions (Figure 2.15). So this operation can be naturally applied to sequence data. In contrast to an RNN, a CNN learns a fixed-length local context and has control of the length of dependencies to model by adjusting kernel size.

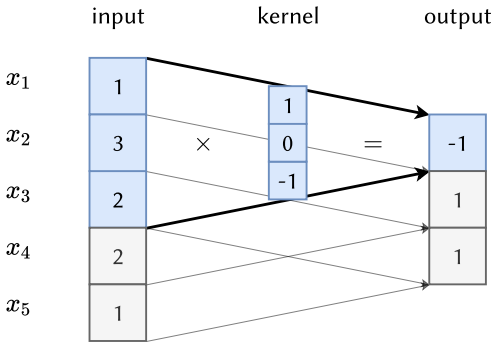


Figure 2.15: An illustration of convolution operation over 1-dimensional data. The operation is essentially a dot product of a kernel vector and a slice of the sequence.

Figure 2.16 shows an example of applying a CNN for sentence classification proposed by Gehring et al. [72]. A word is represented as a d-dimensional vector and a sequence $x_1, x_2, \ldots, x_{t_x}$ can be interpreted as 1-dimensional data with d channels. Three kernels with different colors are shown in the figure. Each kernel has a window size $k = 3$. After convolution and pooling, the original

sequence is mapped into a vector space which can be processed by a standard feed-forward network.

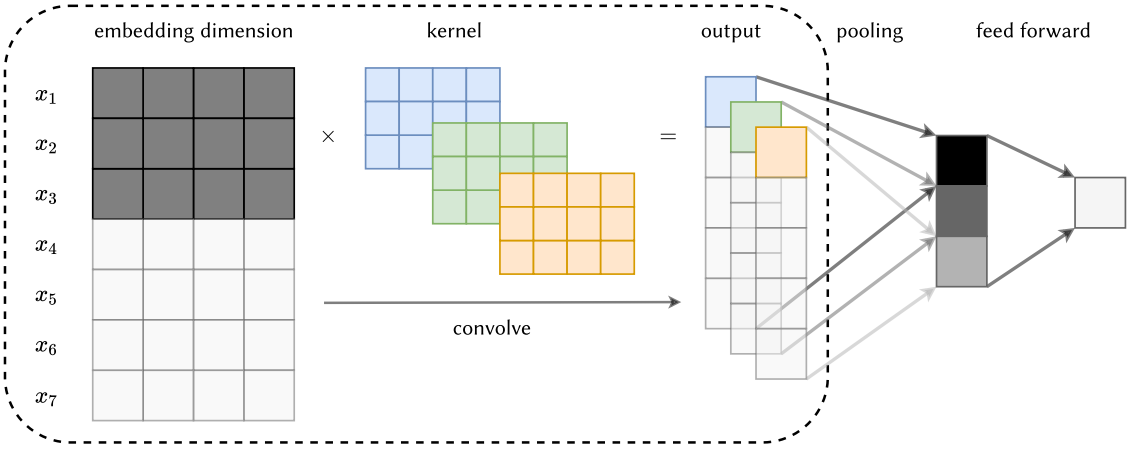


Figure 2.16: CNN for sequence modeling.

Using CNN for sequence modeling is an attempt prior to Transformers in reducing the sequential dependencies of RNN as the convolution calculation can be fully parallelized. More recently, some new connections between CNN and Transformers were unveiled by Cordonnier et al. [41]. They inspected the behaviors of CNN and Transformer vision tasks and found that the self-attention mechanism is at least as expressive as any convolution layer. More generally, self-attention can behave very similarly to convolution on local contexts.

Seq2seq. The seq2seq model is a type of sequence model whose input and output are both sequence data. It was originally proposed by Sutskever et al. [194] for machine translation but was shortly adopted for a variety of text-related tasks including text summarization, question answering, and the focus of this chapter – query generation. We will use the term *seq2seq* to refer to these tasks and all related models more generally.

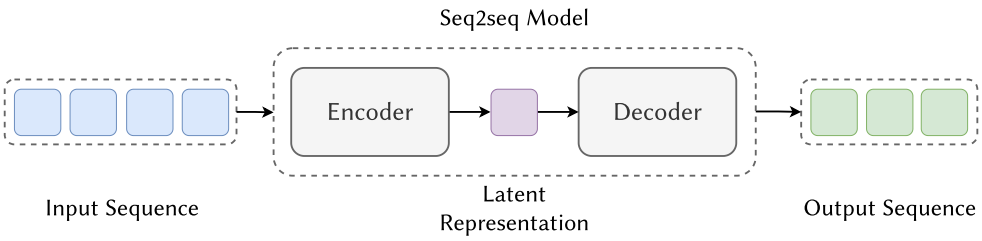


Figure 2.17: Seq2seq abstraction.

The seq2seq abstraction is illustrated in Figure 2.17. A seq2seq model consists of two parts: an encoder and a decoder. The encoder receives the input sequence and encodes it into a latent representation; the decoder decodes the latent representation into the output sequence. The encoder-decoder design relaxes the length alignment constraints between the input and the output, so the model can produce a sequence of arbitrary length which is often the case for many tasks such as machine translation.

Typical seq2seq models include RNN [177], LSTM [81], and Transformer [201]. Figure 2.18 shows how a seq2seq model is implemented using RNN. The input sequence is encoded into h_4 using one RNN cell. Then another RNN cell yields a new word $\hat{y}_1$ based on h_4 and continue generating new words using previously generated words. To optimize such a model, we calculate the cross entropy loss which is equivalent to maximum likelihood estimation given the ground truth sequence $Y = y_1, y_2, \ldots, y_{t_y}$:

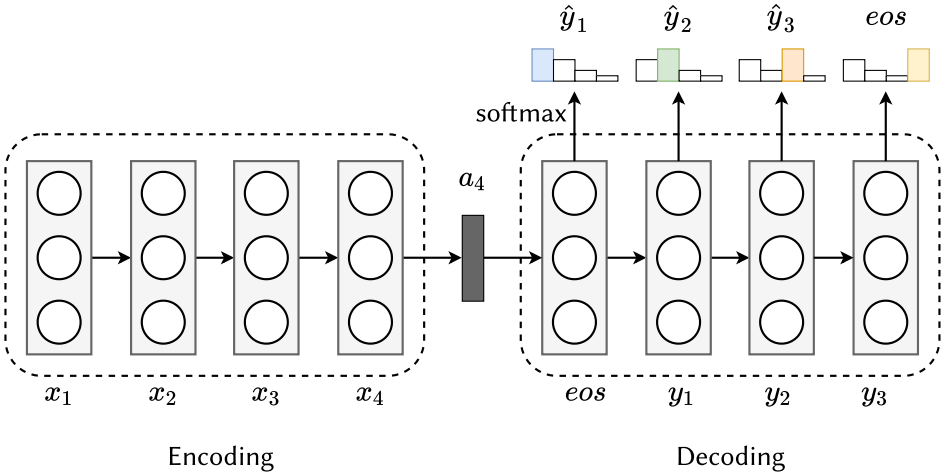
$$L = -\sum_{t=1}^{t_y} \log P(y_t | \hat{y}_1, \hat{y}_2, \cdots, \hat{y}_{t-1}, X) \tag{2.31}$$

where $P(y_t|\cdot)$ is the probability of the word y_t produced by a softmax operation. Inference works a bit differently from training. The model needs to condition on the actual generations in generating the next word. During generation, the output is a distribution over the entire vocabulary, so there are two ways of choosing words. The *greedy* approach chooses the word with the highest probability at each generation step. However, this approach often leads to low-quality results as local optimums do not guarantee global optimum. To get an optimal overall output

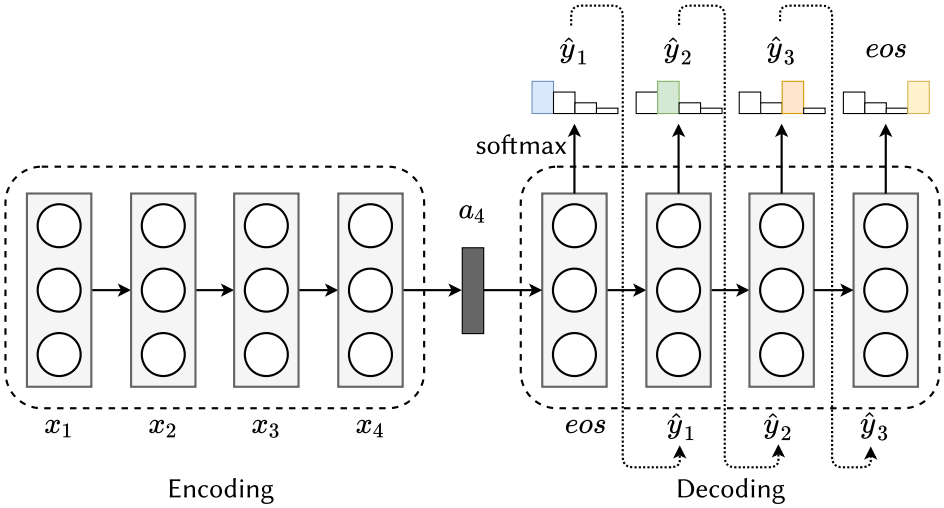
$$\arg\max_{\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_t} P(\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_t | X),$$

we need to examine the probability of every possible combinations of words. For example, to get the optimal sequence of n words, the number of candidate sequences to examine is v^n where v is the vocabulary size, which is infeasible in practice. The *beam search* algorithm is a trade-off and is widely used. It maintains k sequences of words during generation. At each time step, k sequences becomes $k \times v$ sequences. Then only the top k sequences are kept, and the reset are discarded to reduce memory consumption. The process continues until all k sequences end. The *beam search* also does not guarantee a global optimal solution but can significantly improve generation quality in practice.

Now let us look at training again. The output-as-input process can result in problems. If the first prediction $\hat{y}_1$ is wrong, then $\hat{y}_2$ is probably wrong, then $\hat{y}_3$ is also wrong, and finally the entire sequence is wrong. This problem will eventually lead to slower convergence and an unstable model. It can be alleviated using the *teacher forcing* [210] algorithm which uses ground truth tokens as decoding input. That is, y_t is conditioned on $y_1, y_2, \ldots, y_{t-1}$ instead of the model



(a) RNN seq2seq model during training. Typically, the teacher forcing algorithm is used to avoid accumulating errors and the ground truth sequence is used as the input for the decoder.



(b) RNN seq2seq model during inference. The decoding process conditions on real-time generations.

Figure 2.18: RNN Seq2seq model. The input sequence is encoded into a latent representation then decoded into the output sequence. A special *eos* token is used to initiate the decoding process.

prediction $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{t-1}$. The loss for teacher forcing is defined as follows.

$$L = -\sum_{t=1}^{t_y} \log P(y_t | y_1, y_2, \cdots, y_{t-1}, X) \tag{2.32}$$

Unsurprisingly, the generation quality of a seq2seq model depends on the encoder to compress all the information of the input sequence into a fixed-length vector (h_4 in Figure 2.18). While this might be sufficient for short sequences, it is often a bottleneck in processing long sequences [6]. Cho et al. [33] show that the translation performance degrades dramatically as the sequence length grows. In the next section, we discuss the *attention* mechanism which allows the decoder not only use hidden information, but also use information from the original sequence.

Attention Mechanism. In order to alleviate the limited expressiveness of representing a sequence with a single vector, Bahdanau et al. [6] first proposed the *attention* mechanism, which imitates human attention in understanding text, for machine translation. The concept of attention is very intuitive. See Figure 2.19 for an example. To understand the sentence, we need to link different words: to understand what is being eaten, "eating" and "apple" are clearly more useful than "eating" and "green". So, we pay more attention to "apple" than "green" when we read "eating". This is the intuition behind attention — words having different importance to each other.

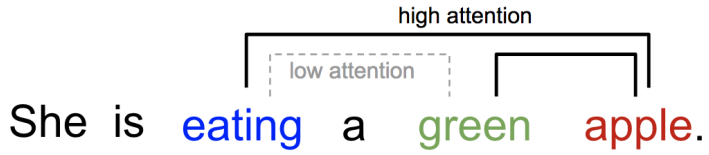


Figure 2.19: The importance of context in a sentence. (Image source: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html)

To formally describe attention, we use the following notation. We have a target sequence $\{t_1, t_2, \ldots, t_m\}$ and a source sequence $\{s_1, s_2, \ldots, s_n\}$ where m and n are the number of words in the target and source sequence correspondingly. Let $\mathbf{t_i} \in \mathbb{R}^d$ and $\mathbf{s_j} \in \mathbb{R}^d$ denote the column vector representation of the word t_i and s_j in a d-dimensional space. Thus we can convert the sequences into vector representations: $T = \{\mathbf{t_1 t_2} \ldots \mathbf{t_m}\}$ and $S = \{\mathbf{s_1 s_2} \ldots \mathbf{s_n}\}$. Given T and S, The attention mechanism assigns an importance weight of every word in S to every word in T.

The source sequence $S = \{\mathbf{s_1 s_2} \ldots \mathbf{s_n}\}$ is first projected into a *key* space $K = \{\mathbf{k_1 k_2} \ldots \mathbf{k_m}\}$ where $\mathbf{k_i} \in \mathbb{R}^d$ and a *value* space $V = \{\mathbf{v_1 v_2} \ldots \mathbf{v_m}\}$ where $\mathbf{v_i} \in \mathbb{R}^d$. The target sequence $T = \{\mathbf{t_1 t_2} \ldots \mathbf{t_m}\}$ is mapped into a *query* space $Q = \{\mathbf{q_1 q_2} \ldots \mathbf{q_m}\}$ where $\mathbf{q_i} \in \mathbb{R}^d$. We generalize the concepts of query, key and value from Vaswani et al. [201] without specifying how the projection is performed. Figuratively, after the projection, the *query* decides its attention by checking the *key* and extracts information from the *value*.

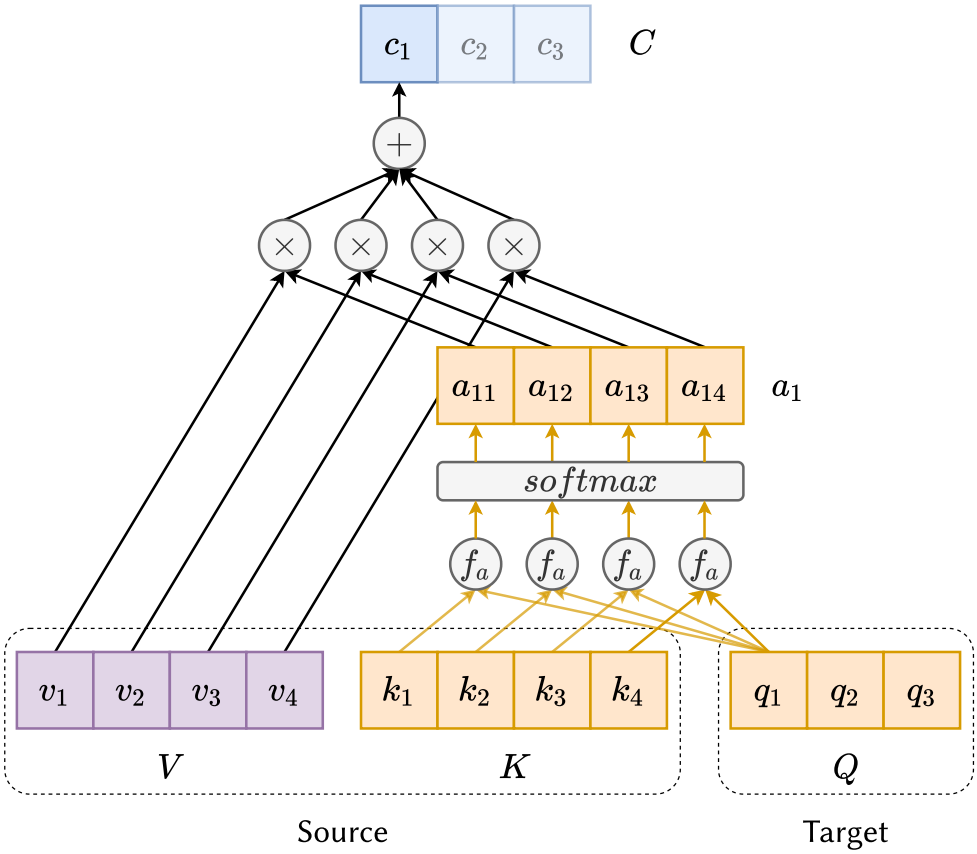


Figure 2.20: A framework for the attention mechanism. Q and K are used to calculate the attention distribution a_{ij} via Eq 2.33 where i denotes the attending word and j denote the attended word. V is then aggregated into a new context vector according to the attention weights via Eq 2.34.

Using Q and K, an attention distribution A is calculated. Let $A = \{\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_m}\}$ and $\mathbf{a_i} = [a_{i1}, a_{i2}, \ldots, a_{in}]$. The attention from the i-th word to the j-th word is defined as follows:

$$a_{ij} = \frac{\exp(f_a(\mathbf{q_i}, \mathbf{k_j}))}{\sum_{j'} \exp(f_a(\mathbf{q_i}, \mathbf{k'_j}))}, \quad i = 1, 2 \ldots, m, \quad j = 1, 2 \ldots, n \tag{2.33}$$

where f_a is a scoring function between two vectors and its definition differs across various attention mechanisms. With the attention weights $\mathbf{a_i}$, V is aggregated into a new context vector $\mathbf{c_i}$.

$$\mathbf{c_i} = \sum_j a_{ij} \mathbf{v_j}, \quad i = 1, 2 \ldots, m, \quad j = 1, 2 \ldots, n \tag{2.34}$$

The entire process is illustrated in Figure 2.20. The specific choice of query, key, value vectors is architecture dependent. Table 2.5 summarizes how these vectors are defined in different architectures. RNN and LSTM based methods use hidden states from the encoder and the decoder while Transformers use a dedicated feed-forward network for the projection. Another major difference is how the score function f_a is defined which is also shown in Table 2.5.

Table 2.5: Common attention mechanisms. $\mathbf{h_{i-1}}$ and $\mathbf{h_j}$ denotes the $(i-1)$-th hidden states in the target sequence and j-th hidden states in the source sequence. $\mathbf{W_q}$, $\mathbf{W_k}$, $\mathbf{W_v}$, $\mathbf{W_a}$ and $\mathbf{v_a}$ denote trainable parameters. d is the dimension of the query and key vectors.

Name	Architecture	$\mathbf{q_i}$	$\mathbf{k_j}$	$\mathbf{v_j}$	f_a
Additive [6]	RNN	$\mathbf{h_{i-1}}$	$\mathbf{h_j}$	$\mathbf{h_j}$	$\mathbf{v_a}\tanh(\mathbf{W_a}\left[\mathbf{q_i} \oplus \mathbf{k_j}\right])$
General [124]	LSTM	$\mathbf{h_{i-1}}$	$\mathbf{h_j}$	$\mathbf{h_j}$	$\mathbf{q_i}^\top \mathbf{W_a} \mathbf{k_j}$
Location only [124]	LSTM	$\mathbf{h_{i-1}}$	$\mathbf{h_j}$	$\mathbf{h_j}$	$\mathbf{W_a}\mathbf{q_i}$
Dot product [124]	LSTM	$\mathbf{h_{i-1}}$	$\mathbf{h_j}$	$\mathbf{h_j}$	$\mathbf{q_i}\mathbf{k_j}$
Scaled dot product [201]	Transformer	$\mathbf{W_q t_i}$	$\mathbf{W_k s_j}$	$\mathbf{W_v s_j}$	$\frac{1}{\sqrt{d}}\mathbf{q_i}\mathbf{k_j}$

There is a special case of attention called *self-attention* recently proposed by Lin et al. [115] where the target sequence and the source sequence are the same. In order to distinguish self-attention from the attention between two sequences, we refer to the latter as *cross-attention*. Self-attention is often used to learn contextualized sentence embeddings. Due to the rich representations self-attention can learn, models with self-attention have performed exceptionally well on sequence classification tasks such as sentimental analysis and textual entailment, which is beneficial to IR where relevance judgment can also be viewed as classifying relevant or non-relevant.

2.4.2 Transformer

As we mentioned previously, early sequence models have several limitations: 1. the expressiveness of using a single vector to represent the sequence; 2. the sequential dependency in processing sequence elements. Transformer architectures have addressed the first issue by using self-attention and the second issue by processing sequence elements simultaneously. Using self-attention leaves no room for information loss even when processing long sequences. Using positional embeddings completely removes the sequential dependency while retaining dependency information.

The self-attention in a Transformer is the same as the attention mechanism we described in Sec 2.4.1 but has a focus on looking at the sequence as a whole. Specifically, the m-word target sequence $\{t_1, t_2, \ldots, t_m\}$ and the n-word source sequence $\{s_1, s_2, \ldots, s_n\}$ are now represented as matrices $\mathbf{T} \in \mathbb{R}^{m \times d}$, $\mathbf{S} \in \mathbb{R}^{n \times d}$ (d is the embedding dimension) instead of two sequences of vectors. This is a trivial change in perspective, but it implies important efficiency improvements. Importantly, a key in removing sequential dependencies is the *positional embeddings*, which en-

code positional information to account for word ordering. Every word embedding vector is added with a positional embedding vector before being processed by the attention mechanism. Positional embeddings only depend on the position of a word in a sequence, and can be learned or fixed. Vaswani et al. [201] originally proposed using sinusoid for positional embeddings. At each position, two signals are interleaved according to Eq 2.35. Figure 2.21 shows the values of 5 positions, each of which contain 512 dimensions. Note that a curve at odd number positions is interleaved with a curve at even number positions.

$$positional\_embedding(pos, dim) = \begin{cases} \sin(pos/10000^{dim/d}) & \text{if } dim = 0, 2, \ldots \\ \cos(pos/10000^{(dim-1)/d}) & \text{if } dim = 1, 3, \ldots \end{cases} \tag{2.35}$$

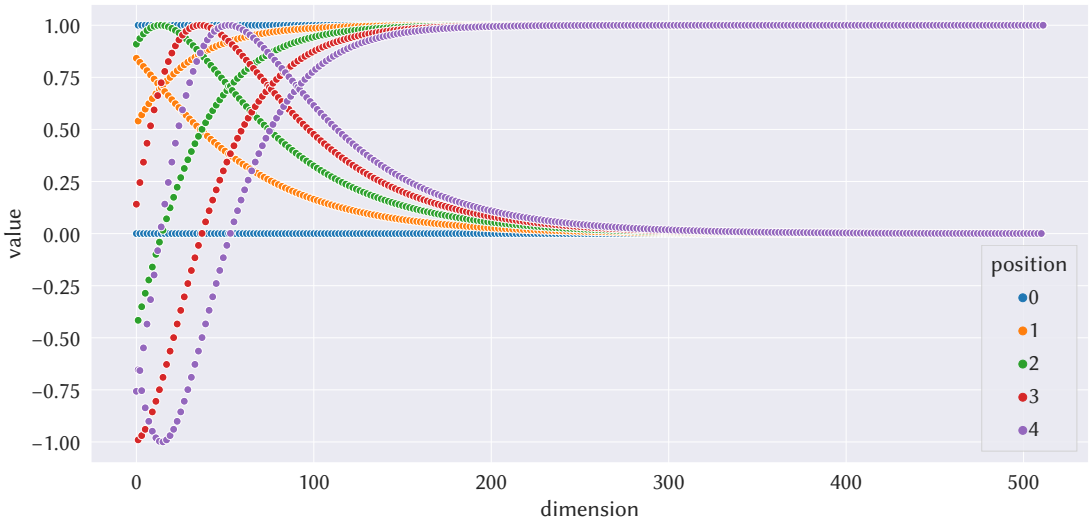


Figure 2.21: Sinusoidal positional embeddings proposed by Vaswani et al. [201].

Let $\mathbf{P}_T$ and $\mathbf{P}_S$ denote the positional embedding matrices which can be two slices of the same embedding matrix. $\mathbf{T}$ and $\mathbf{S}$ are then updated by adding the positional embeddings:

$$\underset{m\times d}{\mathbf{T}} = \underset{m\times d}{\mathbf{T}} + \underset{m\times d}{\mathbf{P}_T}$$

$$\underset{n\times d}{\mathbf{S}} = \underset{n\times d}{\mathbf{S}} + \underset{n\times d}{\mathbf{P}_S}$$

Note that we use the notation

$$\underset{m\times d}{\mathbf{T}} : \mathbf{T} \in \mathbb{R}^{m\times d}$$

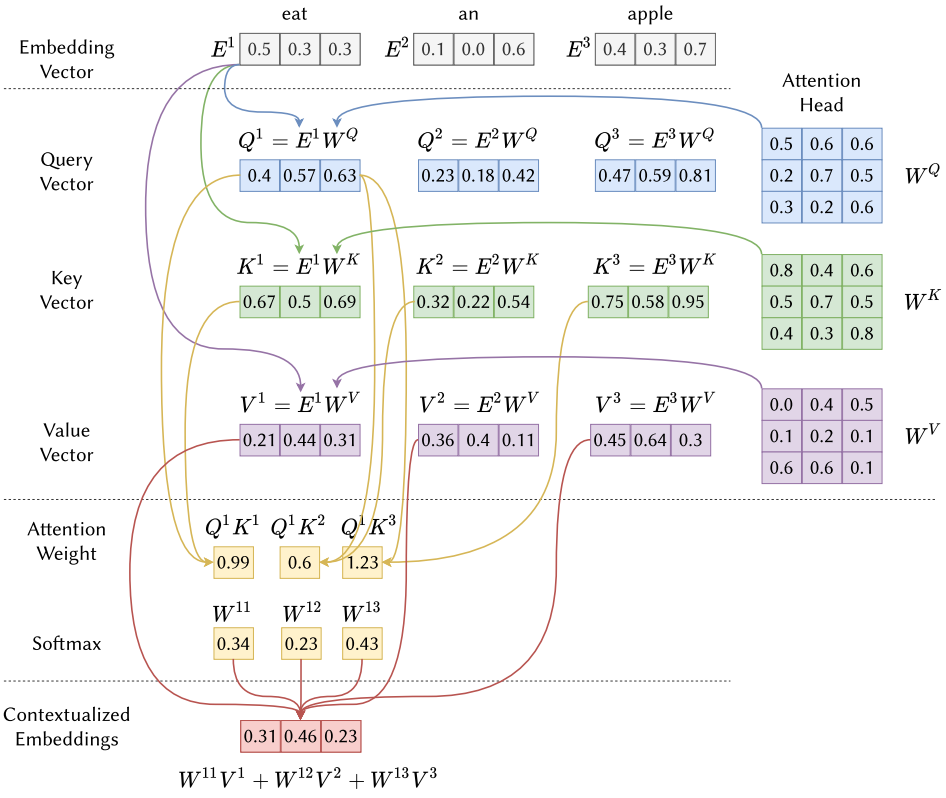


Figure 2.22: An example of self-attention in transformers. The embeddings of "eat" is updated with the embeddings of "eat", "an", "apple" according an attention distribution.

to annotate the matrix with it shape to improve readability. Now we have full information encoded in $\mathbf{T}$ and $\mathbf{S}$. They are converted into query, key, and value matrices:

$$\underset{m\times d}{\mathbf{Q}} = \underset{m\times d}{\mathbf{T}} \times \underset{d\times d}{\mathbf{W}}$$
$$\underset{n\times d}{\mathbf{K}} = \underset{n\times d}{\mathbf{T}} \times \underset{d\times d}{\mathbf{W}}$$
$$\underset{n\times d}{\mathbf{V}} = \underset{n\times d}{\mathbf{S}} \times \underset{d\times d}{\mathbf{W}}$$

Accordingly, the query, key, value vectors can also be represented using matrices such that $\mathbf{Q} \in \mathbb{R}^{m\times d}$, $\mathbf{K} \in \mathbb{R}^{n\times d}$, $\mathbf{V} \in \mathbb{R}^{n\times d}$. The self-attention in Transformers can be simply described by:

$$\underset{m\times d}{\mathbf{C}} = \text{softmax}(\frac{1}{\sqrt{d}} \underset{m\times d}{\mathbf{Q}} \underset{d\times n}{\mathbf{K}^\top}) \underset{n\times d}{\mathbf{V}} \tag{2.36}$$

where a softmax is applied to the last dimension of the matrix. The computation is fully parallelized across the sequence (the m and n dimension of the matrices). Note that $\mathbf{C} \in \mathbb{R}^{m\times d}$ has

the same shape as $\mathbf{T} \in \mathbb{R}^{m \times d}$, which means this operation can be easily repeated multiple times without extra projecting and reshaping which facilitates stacking up multiple attention layers for larger and more powerful models.

Figure 2.22 shows a concrete example of how self-attention is calculated inside a transformer for the term "eat". All the terms are first represented as embedding vectors (**E**). Embedding vectors are then mapped to query (**Q**), key (**K**), and value (**V**) vectors. Multiplying query vectors and key vectors results in attention distribution (**W**). The dot product of the attention distribution and value vectors results in the final contextualized embeddings of "eat".

Note that a transformer is also a seq2seq model (Sec 2.4.1), which means it has an encoder and a decoder. A key difference in an encoder and a decoder is how dependencies are modeled which is an important inspiration for our work in Chapter 5. We will revisit attention again from the perspective of dependency modeling in Sec 5.2.3.

2.4.3 Neural Language Models

The success of Transformers in various text-related fields are not only derived from the architecture but also from pretraining. Traditionally, the parameters of a neural network are randomly initialized and trained for specific tasks. However, as the number of parameters grows, the time and cost of training a network become higher and can become prohibitive. Pretraining, with the idea that general knowledge should be transferable across related tasks, has significantly relieved the burden of training a new network from scratch every time. With a pretrained model, downstream tasks then only need slight finetuning to achieve competitive results. The pretraining-finetuning paradigm is now regularly adopted in designing large neural models, which makes powerful models more accessible and benefits numerous real-world tasks.

The tasks used for pretraining should be general so that the learned knowledge can be widely applied and easily transferable. *Language modeling* thus has been playing a central role in pretraining as it learns generic language knowledge. Let us briefly review language models first. In Sec 2.2.5 and Sec 2.2.6, we have covered n-gram language models and their use in the query likelihood model and sequential dependency model. In short, a language model estimates a probability of a sequence of words $P(w_1, \ldots, w_m)$. Traditional language models use statistics (mostly word frequency) to estimate the probability and are referred to as *statistical language models* [223] in the literature. It is known that statistical language models suffer from the *curse of dimensionality* and have inherent difficulties in capturing long-term dependencies and generalizing across contexts [73]. Concretely, the number of possible n-grams of a vocabulary sized v is v^n which grows exponentially as the dependency range n grows. This often causes a severe data sparsity problem as many possible n-grams may not be observed in training. Although it can to be alleviated by smoothing techniques such as Dirichlet smoothing and Jelinek-Mercer smoothing, the final performance is sensitive to smoothing and far from being optimal. Another problem is statistical models are hard to generalize since words are represented as discrete symbols. Vocabulary mismatches are a representation of this problem. "teenage pregnancy" and "teen pregnancy" are two different queries and can show significant difference in retrieving performance.

Using neural networks to approximate language models have alleviated the aforementioned problems. In contrast to statistical models, these models are referred to as *neural language models* in the literature [16]. They have been shown to be rich in general linguistic knowledge [35] and have properties beneficial to many text-related fields including IR. Neural language models can scale well by using neural networks to approximate language probabilities. The number of parameters does not grow as the range of dependencies grow, so long-term dependencies can be relatively easily modeled. They also generalize well by representing words in a continuous space which is a nice property for relieving vocabulary mismatches. In a properly trained language model, "king"−"man" + "woman" results in a vector closest to "queen" [130] in the latent space. It is no surprise that "teenage" and "teen" are also very close to each other in such a continuous space. Beyond these theoretical advantages, the empirical success of neural language models has also been observed by numerous work reporting remarkable progress in a variety of natural language tasks. Language model pretraining is now routinely applied in IR: using a finetuned transformer ranking model to maximize ranking effectiveness after an initial filtering stage using a retrieval method such as BM25 is now a common practice. For example a large portion of models follow this paradigm in the recent TREC-COVID track [203].

BERT. Among the wealth of pretrained transformer models, BERT [57] has arguably been one of the most successful adapted to the ranking task. BERT uses only the encoder component of a transformer architecture and was pretrained with a *masked language modeling* (MLM) objective. The traditional language models aim at predicting the *next* word given a sequence of words and is also called *causal language modeling* (CLM) due to the use of sequential dependencies. In contrast, the MLM predicts a few masked words in the sequence without being limited to conditioning on the past words (Figure 2.23a). Formally, let $S = \{w_1, w_2, \ldots, w_m\}$ denote a sequence of words; let $M \subset S$ denote some randomly chosen words that are masked; let U denote the original sequence with words in M being replaced with a special "[mask]" token. The objective of MLM is to maximize the likelihood to successfully recover M, or minimize the following loss:

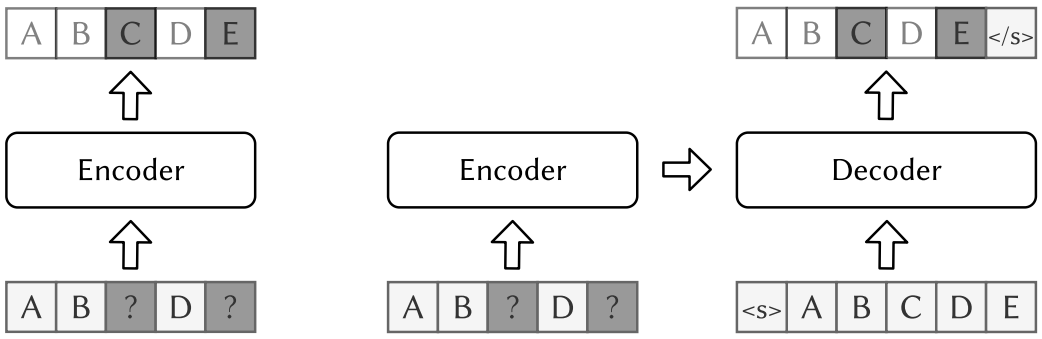
$$L_{MLM} = -\sum_{w \in M} \log P(w|U) \tag{2.37}$$

BERT uses a bidirectional attention mechanism conditioned on U which learns relationships between words and contexts. BERT based models have consistently produced competitive approaches for IR tasks such as MS-MARCO passage (according to disclosed models) and document ranking [143], TREC 2019 [44], and CAsT 2019 [54], and generally require only a moderate amount of fine-tuning.

Fine-tuning BERT is straight-forward. As discussed in Sec 2.3.3, the training of a neural ranking model is orthogonal to the model architecture. Thus, a transformer ranking model can be trained the same way as other neural ranking models. Guo et al. [76]'s abstraction for ranking models (Equation 2.20) still applies. Under their framework, ψ and ϕ are null functions, η simply looks up word embeddings given word indexes, and g is the entire network including the atten-

tion mechanism. For example, Nogueira and Cho [145] directly apply the fine-tuning approach described by Devlin et al. [57] to produce a competitive system. They cast ranking as sentence pair classification and fine-tune BERT using the MS-MARCO training data. Dai and Callan [51] employ the same fine-tuning approach and observed significant improvements on other commonly used IR test collections such as Robust04 and ClueWeb09B. In more recent work, Nogueira et al. [147] proposed a multi-stage re-ranking system with a modified fine-tuning approach.

BART. BART is a pretrained encoder-decoder transformer. The core idea is to corrupt data (e.g. mask words, remove words, change word orders) and train the model to recover the data. Figure 2.23b shows how masked language modeling is performed on BERT and BART. On the encoder side, the input sequences are the same as in BERT while on the decoder side, sequential dependencies are retained. Nogueira et al. [149] explore the use of an encoder-decoder transformer T5 [164] for ranking. By casting relevance prediction as text generation, the model is trained to predict "True" or "False" literals given a query-document pair. The documents are then re-ranked based on the probability assigned to "True" token. However, applying encoder-decoder models generally require modifications to model training and incur additional retraining costs as the models tend to be much larger than an encoder-only architecture. It is also unclear exactly how an encoder-only architecture and an encoder-decoder architecture differ in terms of retrieval effectiveness. We include a detailed study in Section 5.5.4 that explores this question further.



(a) BERT pretraining.

(b) BART pretraining.

Figure 2.23: Language model pretraining. In a sequence of words "ABCDE", "C" and "E" are masked out and represented with "?". Models are trained to predict these masked words by learning to infer from the context. "<s>" and "</s>" are special tokens indicating the beginning and ending of a sequence. The sequences on the decoder side are shifted to maintain the sequential dependency.

2.4.4 Sequence Models for IR

Sequence modeling has long been used throughout IR. We summarize the representative work in Table 2.6. First, each work has a focus on learning representation, end-to-end ranking, and/or generation. Second, each work can also be classified according to the objective: ranking effectiveness and/or readability. The readability objective is the maximum likelihood objective. Since the targets are human-written text we denote it as readability.

Early models focused on learning representations for queries and documents and use simple functions such as cosine similarity to measure relevance. Later, there was a shift towards using sequence models for end-to-end ranking as it has been observed that query-document interactions are crucial in deciding relevance. Early models are rarely used for generation which is particularly difficult, from both research and engineering perspectives. From the research perspective, generation has a challenge in text understanding especially in modeling of long-term text dependencies. This was not made easy until attention was invented. The inefficiency caused by sequential dependencies was also nontrivial to relieve. From an engineering perspective, transformers are not only a novel architecture but also an engineering masterpiece. Transformers have defined a set of interfaces for sequence modeling which flattened the learning curve. Sharing the same functional interface also makes numerous transformer-based models readily available. The parallelized internal calculation can also fully exploit the computational power of modern GPUs. Since the emergence of transformers, researchers have easily accessible tools [211] to explore generation tasks for IR while ranking remains the main focus.

There was not a dominant architecture in early research due to the mixed results reported from those models. While none was dominant, researchers put an emphasis on designing their own networks. Again, as transformer emerged, it has become prevalent due to its ability of capturing long-term dependencies and fully parallelized computation.

Since early work focused on ranking, they were mainly trained with ranking effectiveness objectives. Some recent work focused on generating queries but not with an effectiveness objective. Chapter 4 (SNLQ) focuses on both effectiveness and readability of generated queries. Chap 5 (GDMTL) jointly models ranking and generation under a unified framework.

2.5 Reinforcement Learning

Reinforcement learning is a machine learning paradigm that enables a learning agent to learn by interacting with an environment. The learning process is driven by rewards the agent receives from the interactions, which is analogous to how human-beings learn: a player learns by playing games and winning and losing. More formally, this interaction loop is illustrated in Figure 2.24. At time step t, the agent observes the environment, and then based on the observed *state* S_t, the agent issues an *action* A_t to the environment. The environment provides a *reward* R_t to the agent and changes its state to S_{t+1} as a result of the interaction. After receiving a reward, the agent then adjusts itself accordingly. The entire process can be represented as a sequence: $S_0, A_0, R_0, S_1, A_1, R_1, \ldots$. The learning objective is for the agent to maximize the *return* G_t, defined

Table 2.6: Representative sequence models for IR. The main focuses of using sequence models for IR are learning representations (Rep), end-to-end ranking (Rank), and generation (Gen). They can be trained with ranking effectiveness or readability (maximum likelihood) objectives.

	Year	Arch	Focus			Objective	
			Rep	Rank	Gen	Effective	Read
DSSM [85]	2013	FFNN	✓			✓	
CDSSM [188]	2014	CNN	✓			✓	
R2N2 [31]	2015	RNN		✓		✓	
DRMM [75]	2016	FFNN	✓	✓		✓	
Duet [132]	2017	CNN		✓		✓	
DeepRank [155]	2017	CNN+RNN		✓		✓	
K-NRM [215]	2017	FFNN		✓		✓	
SNRM [219]	2018	FFNN	✓	✓		✓	
CONV-KNRM [52]	2018	CNN		✓		✓	
BERT [51]	2019	Transformer		✓		✓	
BERT [145]	2019	Transformer		✓		✓	
Doc2query [148]	2019	Transformer			✓		✓
SNLQ [116]	2020	Transformer			✓	✓	✓
GDMTL [117]	2021	Transformer		✓	✓	✓	✓

as a long-term accumulated reward:

$$\begin{aligned} G_t &= R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k} \end{aligned} \tag{2.38}$$

where γ is the *discounting rate* which discounts future rewards. It is critical that when the agent learns, it always considers the long-term accumulated reward G_t instead of the short-term immediate reward R_t. In a game, we want to use strategies to get the final victory instead of focusing just on short-term benefits.

How to optimize such a model is the key question. The learning algorithms can be divided into two categories: value-based and policy-based. Value-based methods estimate the *value* of a state $V(S_t) = E[G_t|S_t]$: the expected return one can possibly achieve from state S_t onwards, and the *value* of an action under the state $Q(S_t, A_t) = E[G_t|S_t, A_t]$: the expected return one can possibly achieve by issuing action A_t under state S_t. The biggest drawback of this category of algorithms is that the number of possible actions is too large, and so value-based methods usually do not scale well. Unfortunately, this is often possible for text generation where the generation vocabulary is the action space and often contains tens of thousands candidates. Policy-based methods address this issue by estimating a preference for all the actions directly, bypassing the

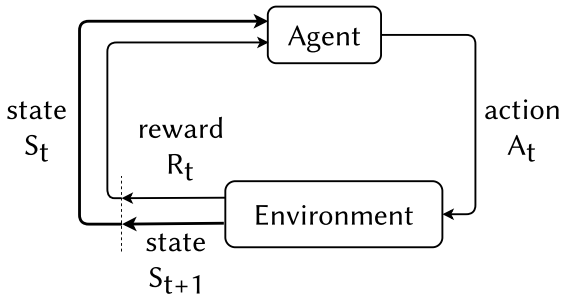


Figure 2.24: The interactions between an agent and an environment. The agent observes the environment and issues an action to interact with the environment, then receives a reward from the environment.

estimation of single action values. So, policy-based methods are more commonly used for seq2seq problems.

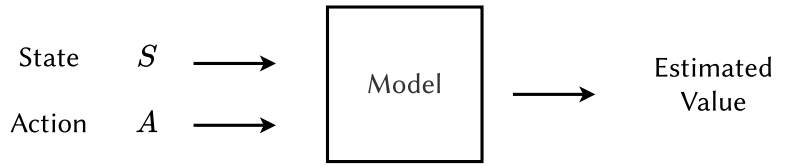
Figure 2.25 highlights the differences in model input and output for value-based and policy-based methods. Value-based methods produce an estimated *value* of state-action pairs while policy-based methods produce a preference over the entire action space.

Policy-Based Methods. A policy π is a parameterized decision-making rule and is often implemented as a neural network. The policy is optimized through *gradient ascent*, so policy-based methods are also referred to as *policy-gradient* methods. When a Policy Gradient is used to optimize neural networks, the rewards can be any value and are not required to be differentiable with respect to network parameters. This property allows us to use a variety of non-differentiable rewards such as summarization metrics like ROUGE, and ranking metrics like MAP and NDCG. This is achieved using the Policy Gradient Theorem

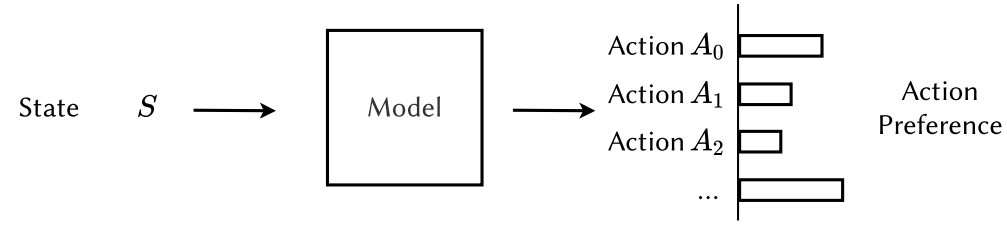
$$\begin{aligned} J(\theta) &= V(S_0) \\ \nabla J(\theta) &= \nabla V(S_0) \\ &\propto E_{S,A\sim\pi}[Q(S,A)\nabla_\theta \ln \pi(A|S)] \end{aligned} \tag{2.39}$$

where $J(\theta)$ is the objective function parameterized by θ, $Q(S,A)$ is the *value* of taking action A at state S, $\pi(A|S)$ is the probability of taking action A in state S. The Policy Gradient Theorem allows us to optimize a policy by simply maximizing the multiplication of the action's *value* and its log likelihood value. In practice, the implementation is much simpler as we can turn the maximization into minimization as in Eq 2.40.

$$-\nabla_\theta J(\theta) \propto E_\pi[Q(S,A)\nabla_\theta - \ln \pi(A|S)] \tag{2.40}$$



(a) Value-based method



(b) Policy-based method

Figure 2.25: An illustration of model designs in value-based and policy-based reinforcement learning methods. Value-based method takes as input an observation of the environment (such as a raw bitmap from a game display) and an action (such as pressing a button on a game controller), and outputs an estimated *value*. In contrast, policy-based methods takes only the environment observation as input, and produce a preference over all the actions.

Observe that $-\ln \pi(A|S)$ is identical to the cross entropy loss if we imagine the action a as the "ground truth". In summary, with the Policy Gradient Theorem, we simply need to sample an action, estimate its *value*, multiply that *value* $Q(S, A)$ by the action's cross entropy loss $-\ln \pi(A|S)$, and minimize the loss. Depending on how $Q(S, A)$ is estimated, Policy Gradient has many variations. Schulman et al. [184] for example provides a summary of commonly used Policy Gradient algorithms.

For seq2seq tasks, the most commonly used algorithm is called REINFORCE [209]. The basic form of REINFORCE uses G_t to estimate $Q(S, A)$. Based on Eq 2.40, the loss of reinforcement learning for seq2seq tasks has the form as Eq 2.41.

$$L = -\sum_{t=1}^{n} \log\ P(\hat{y}_t|\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_{t-1}, x) \cdot R(\hat{y}) \tag{2.41}$$

where $\hat{y}$ denotes the output and $R(\hat{y})$ denotes the task-dependent reward, such as ROUGE [114] or BLEU [156] scores against the targets.

2.6 Summary

In Sec 2.1, we introduced a typical workflow of an information retrieval process, including indexing, query optimization, multi-stage retrieval, and evaluation. We also put the content of the thesis in the context of the process. In Sec 2.2, we introduced traditional retrieval models. Particularly, the language modeling approach is the foundation of our fielded relevance models. In Sec 2.3 we introduced a neural ranking framework and the development of neural ranking models. We also introduced key techniques of training neural ranking models. In Sec 2.4, we introduced sequence models, including RNN, CNN and Transformers. We also discussed several shortcomings of early models such as representation bottlenecks and inefficiency, and how tackling these shortcomings lead to the invention of the attention mechanism and transformers. At the end of the section, we also discussed the application of sequence models in IR. Finally, in Sec 2.5, we introduced the principles of reinforcement learning and how it works in optimizing IR objectives when combined with seq2seq models.

3 Optimizing Queries with Field-Based Relevance Models

3.1 Introduction

Query reformulation, query expansion, query re-weighting, and query generation are all considered to be *query optimization* techniques which aim at improving the representations of the underlying *information need*. Sometimes slightly reformulating or expanding a query can lead to dramatic performance improvements if the new query manages to address potential vocabulary mismatches or more verbosely specifies the information need. For instance, in one preliminary experiment, we found that reformulating "teenage pregnancy" to "teen pregnancy" improves the Mean Average Precision (MAP) from 0.086 to 0.261 for the TREC Robust04 2004 (TREC disks 4 and 5) collection [205]. Expanding the query to "teen pregnancy prevention" further improves the score to 0.386. It is impossible for a user to know if a query is the best for a particular data collection, and query optimization techniques can be used to alleviate such vocabulary mismatches or under-specification problems.

One of the most fundamental and principled paradigms to query optimization is *relevance models* [106] where terms in the query and in relevant documents are assumed to be generated by a latent relevance-based language model. A relevance model is usually induced from pseudo relevant documents — i.e., those most highly ranked by an initial search. We consider relevance models as a general query optimization technique as they can be used for either reformulating (RM1 [106]), expanding (RM3 [1]), re-weighting [106], or generating [18] a query.

However, past work on relevance models has focused on unstructured text. When inducing a relevance model from a structured document consisting of multiple *fields*, e.g. *title*, *heading*, and *inlink* which are derived from HTML and SGML markup, important signals in document structures may be lost as every document is treated as a bag of words and every field is considered equally important. Empirically, the match between a query and document fields is often assumed to be a strong relevance signal. For example, titles and headings are usually more concise and descriptive than the main text of a document. Figure 3.1 shows an example of the structure of the *Information Retrieval* wikipedia article. The headings of the article provide important information

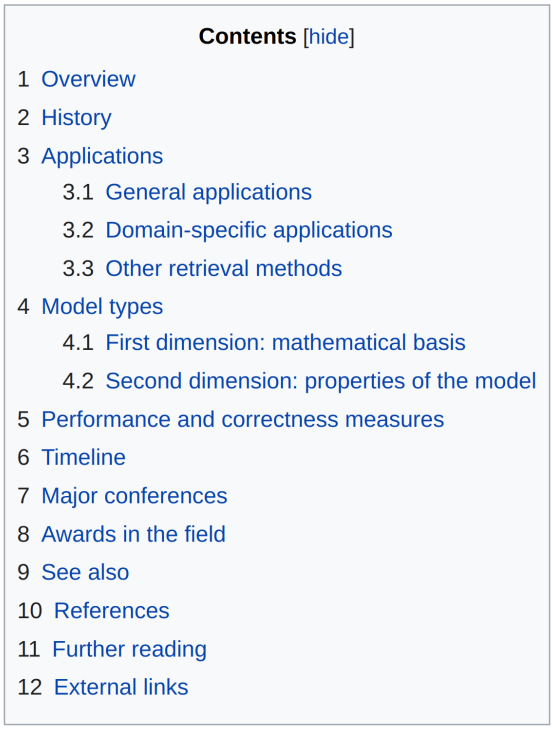


Figure 3.1: The structure of the *Information Retrieval* wikipedia page.

about the content of the article and can be used to surface relevance signals. Commonly used Web retrieval methods such as BM25F [221] and FSDM [129] outperform their non-field counterparts. From a theoretical point of view, traditional relevance models assume that the entire document is generated from the same language model. This may not be true as can be observed from the field term frequency analysis in Figure 3.2. Different fields share a high-level general trend, but the word preferences may vary dramatically.

We first present a study of using field-based information in the relevance modeling framework. Our first method induces relevance models from fields independently and then linearly combines them to create a weighted model. The second method is based on inducing a relevance model from the entire document and using it to score fields. Hence, fields are used in two (integrated) capacities: sources of information for inducing relevance models and units scored by using relevance models. Another important aspect of this chapter is a comprehensive failure analysis of field-based retrieval performance when applying relevance modeling. While past work has demonstrated the *average* effectiveness of field-based methods, failure analyses are rare. These are important for shedding light on potential avenues for performance improvement.

Experiments performed using ClueWeb09B collection show that while common field-based ranking models improve early precision effectiveness, using field-based information with relevance models can further improve it, specifically, with respect to using relevance modeling with

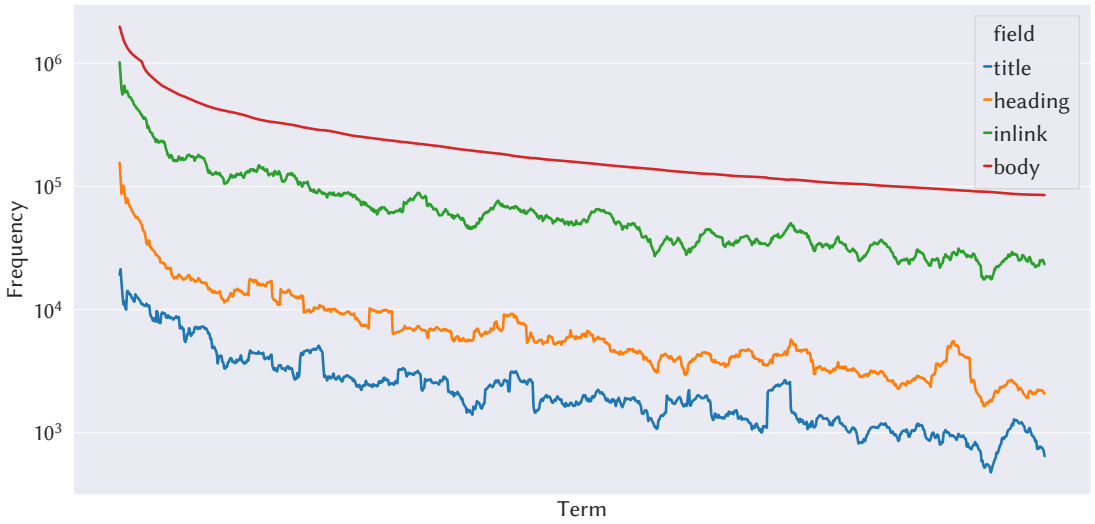


Figure 3.2: Field term frequencies of web documents in ClueWeb09B. Terms are ordered by the frequency in the document body (denoted as *body*). *title* is HTML <*title*> tag; *heading* is a collection of <*h1*>, <*h2*>, <*h3*>, and <*h4*> HTML tags; *inlink* is the anchor texts of hyperlinks; *body* is the <*body*> tag.

whole documents as is the standard. Our failure analysis shows that field-based relevance modeling is mainly effective for tail queries. We also show that using field-based information, with or without relevance modeling, has mixed effects in terms of Mean Average Precision (MAP).

However, relevance models are still subject to interpretability and efficiency issues. A relevance model is a weighted bag-of-words and can expand the query to 25, 50, or even 100 terms. For example, "0.0041 tree 0.0040 family 0.0021 gift 0.0020 engrave 0.0008 birthstone 0.0003 designs ..." is a query generated by relevance modeling in our preliminary experiments. This type of query is unreadable to human beings and hard to process for modern neural ranking models such as BERT due to the lack of syntax structures. The length of this type of query also requires massive computational resources to process. These factors motivate us to rethink what a good query is and investigate the prospects of imitating human written queries.

In order to do this, we comprehensively compare sets of automatically generated queries — produced using random walks over click graphs derived from Bing query logs — and human generated queries using two commonly used TREC test collections. This approach differs from previous studies which made comparisons on a query-by-query basis instead of comparing performance distributions of multiple queries for a single information need simultaneously. We show that the retrieval effectiveness of the automatic queries can reach the level of human-created queries, although the two sets of queries (per-topic) are quite different in several respects: the queries themselves, their similarities and the corresponding retrieved lists. So, a performance gap still exists. Promising advances in IR and closely related fields such as natural language pro-

cessing and machine translation may finally make automatic query generation a reality, and we will explore query generation in the next chapter.

3.2 Theoretical Foundation

3.2.1 (Pseudo-)Relevance Feedback for Query Optimization

When different words are used in a query and a document to refer to the same thing, match-based retrieval methods such as Query Likelihood and BM25 may not effectively retrieve the relevant documents. While users can reformulate their queries manually, retrieval systems can also automatically expand/reformulate the query to tackle the problem. Depending on the information used for expansion, Xu and Croft [216] classified automatic query expansions into global methods and local methods. Global methods analyze the entire corpus or employ external thesaurus or query logs [125]. Local methods refer to (pseudo-)relevance feedback which only uses the documents from an initial retrieval list, and are the focus of this chapter.

Relevance Feedback. Relevance feedback asks the user to provide relevance judgments on an initial retrieval list and then refine the retrieval using the feedback. The well-know *Rocchio algorithm* [174] incorporates user feedback into the Vector Space Model (Section 2.2.3). Let $\mathcal{R}$ denotes the relevant set of documents and $\mathcal{N}$ denotes the non-relevant set of documents according to user feedback; $Q = (q_1, q_2, \dots)$ and $D = (d_1, d_2, \dots)$ denote a query feature vector and a document feature vector respectively. So, q_i and d_i refer to the same term in the vocabulary but have different values in Q and D. Every q_i is then updated according the weights of the term in the relevant and non-relevant sets. The mean weight of the term d_i in the $\mathcal{R}$ set is added to q_i; the mean weight of d_i in the $\mathcal{N}$ set is subtracted from q_i:

$$q_i^* = \alpha \cdot q_i + \beta \cdot \frac{1}{|\mathcal{R}|} \sum_{D \in \mathcal{R}} d_i - \gamma \cdot \frac{1}{|\mathcal{N}|} \sum_{D \in \mathcal{N}} d_i \tag{3.1}$$

where α, β and γ are often set to 8, 16 and 4 empirically [46].

This method modifies the original query so that high impact terms in relevant documents will have higher weights and high impact terms in the non-relevant documents will have lower weights. The new query $Q^* = (q_1^*, q_2^*, \dots, q_v^*)$ is then used to calculate the similarity to documents as in Eq 3.2.

$$\begin{aligned} Score_{Rocchio}(Q, D) &= Cosine(Q^*, D) \\ &= \frac{Q^* \cdot D}{\|Q^*\| \cdot \|D\|} \\ &= \frac{\sum_i^v q_i^* \cdot d_i}{\sqrt{\sum_i^v q_i^{*2}} \cdot \sqrt{\sum_i^v d_i^2}} \end{aligned} \tag{3.2}$$

Pseudo Relevance Feedback. In contrast to Relevance Feedback, Pseudo Relevance Feedback does not require users to provide relevance judgments for an initial retrieval list but assumes that the initial retrieved documents are more *likely* to be relevant [106]. It also assumes a query is generated from an underlying relevance-based language model, or simply *relevance model*, $P(w|R)$. This model is estimated from the initial retrieval list and then compared against each document's own language model for ranking.

Since a query is derived from a relevance model, we can estimate the model from the query and then factor $P(w|R)$ into estimating the joint probability of $P(w, Q)$:

$$\begin{aligned} P(w|R) &\approx P(w|Q) \\ &= \frac{P(w,Q)}{P(Q)} \\ &= \frac{P(w,Q)}{\sum_{w'} P(w',Q)} \end{aligned}$$

Assuming that w and Q are independent, $P(w, Q)$ can be further factored as a weighted sum of $P(w|D)$ and $P(Q|D)$:

$$\begin{aligned} P(w,Q) &= \sum_D P(w,Q|D)P(D) \\ &= \sum_D P(w|D)P(Q|D)P(D) \\ &\propto \sum_D P(w|D)P(Q|D) \end{aligned}$$

Note that $P(Q|D)$ is the query likelihood score of D from the initial retrieval list (Section 2.2.5), so this equation can be interpreted as a weighted sum of the probability of w in each document, which for example can be estimated using term frequencies.

The negative KL divergence between $P(\cdot|R)$ and $P(\cdot|D)$ then serves for ranking.

$$\begin{aligned} Score(Q,D)_{RM} &= -KL(P(\cdot|R)\|P(\cdot|D)) \\ &\propto \sum_w P(w|R)\log P(w|D) \end{aligned} \tag{3.3}$$

To alleviate query drift, the original relevance model is anchored to the original query using a free parameter λ, yielding relevance model #3 (RM3) [1]:

$$P_{RM3}(w|R) = \lambda P(w|Q) + (1-\lambda)P(w|R). \tag{3.4}$$

The new relevance model is then used to rank documents according to the KL divergence.

Following Lavrenko and Croft [106], many other techniques have been proposed to improve relevance modeling, as listed in Table 3.1. They are categorized through three perspectives. First, relevance models may be induced from local or external corpora, or even from different fields of structured documents. Second, some researches investigate improving document selection or

Table 3.1: Literature on relevance models. L: Local corpora; E: External source; S: Structured documents. D: Document selection; T: Term selection; Q: Query selection; L: Learned.

Literature	Year	Resources			Strategy			Model
		L	E	S	D	T	Q	L
Relevance Based Language Models [106]	2001	✓						
UMass at TREC 2004: Novelty and HARD [1]	2004	✓						
A Framework for Selective Query Expansion [47]	2004	✓					✓	
Improving the Estimation of Relevance Models Using Large External Corpora [59]	2006		✓					
Latent Concept Expansion Using Markov Random Fields [128]	2007	✓				✓		✓
Estimation and Use of Uncertainty in Pseudo-Relevance Feedback [40]	2007	✓			✓			
Selecting Good Expansion Terms for Pseudo-Relevance Feedback [29]	2008	✓				✓		✓
A Cluster-Based Resampling Method for Pseudo-Relevance Feedback [109]	2008	✓			✓			
Query Dependent Pseudo-Relevance Feedback Based on Wikipedia [217]	2009		✓	✓				✓
Entity Query Feature Expansion Using Knowledge Base Links [53]	2014		✓	✓				
Query Expansion with Freebase [214]	2015		✓	✓		✓		✓
NPRF: A Neural Pseudo Relevance Feedback Framework for Ad-Hoc Information Retrieval [112]	2018	✓						✓

term selection, or deciding which queries should be expanded. Finally, we identify the works combining relevance models and supervised learning techniques.

With regard to the resource of relevance models, most relevance models are derived from local corpora, but the quality of induced models may suffer from low quality documents in these corpora. Diaz and Metzler [59] introduce RM using high quality external corpora as the additional source of relevance models. Later, other researchers extend it to Wikipedia pages [217] and knowledge base [53, 214] by identifying the connection between original queries and entities in those corpora. To further exploit these high quality corpora, Xu et al. [217], Dalton et al. [53] and Xiong and Callan [214] also explore the importance of adjusting terms appearing in different fields. Inspired by their work and other field-based models, In this chapter, we induce relevance models from web document fields directly and observed significant performance improvement. Although relevance modeling is well known for improving the average system per-

formance, some queries may greatly suffer from it. Cronen-Townsend et al. [47] addresses this issue by proposing to apply relevance modeling selectively.

From the second perspective, instead of improving the overall quality of relevance models from external corpora, some work focuses on specific aspects of relevance modeling. Collins-Thompson and Callan [40] and Lee et al. [109] explore improving document selection by sampling and clustering techniques respectively; Cao et al. [29] train a classifier to distinguish good and bad terms; Xiong and Callan [214] devise a term score function to improve the weights of terms.

Finally, we consider the combination of relevance modeling and maching learning techniques. Cao et al. [29] and Xu et al. [217] train a model to classify terms as good or bad, and use only good terms for query expansion. Xiong and Callan [214] use a linear SVM classifier to estimate the probabilities of terms in an expanded query. Li et al. [112]'s work is of special interest as it is the first to adapt neural networks in pseudo-relevance feedback. Their neural network estimates the relevance of a document using pseudo-relevance feedback documents as evidence in addition to the original query.

3.2.2 Field-Based Retrieval

Our focus is on integrating field-based information and relevance modeling. Thus, we briefly review commonly used field-based retrieval methods.

BM25F. BM25F extends BM25 (Section 2.2.4) by incorporating field information directly into the ranking function. Robertson et al. [167] proposed to boost the weights of terms that also appeared in fields. Zaragoza et al. [221] combined the normalized weighted term frequency to produce the BM25F document scoring function:

$$
\begin{aligned}
Score_{BM25F}(Q,D) &= \sum_{t \in Q \cap D} \frac{TF_{pseudo}(D,t)}{k_1 + TF_{pseudo}(D,t)} \cdot IDF(t) \\
TF_{pseudo}(D,t) &= \sum_{f} W_f \cdot \frac{TF_f(D,t)}{1 - B_f + B_f \cdot (\frac{Len_f(D)}{AvgLen_f})} \\
IDF(t) &= \log(\frac{N - DF(t) + 0.5}{DF(t) + 0.5})
\end{aligned}
\tag{3.5}
$$

Same as in BM25, k_1 is a hyperparameter, N is the total number of documents in the collection, $DF(t)$ is the document frequency of t. In addition, this model introduces more field-specific parameters and functions. B_f and W_f are field-specific hyperparameters to control the contributions of different fields, $TF_f(D,t)$ is the term frequency of t in the field f of document D. The length normalization also happens at the field level. $Len_f(D)$ denote the length of field f in document D; $AvgLen_f$ denote the average length of field f in the collection.

BM25F uses *pseudo term frequency* instead of the common term frequency. The pseudo term frequency $TF_{pseudo}(D,t)$ incorporates field information by assigning different weights to the field term frequencies $TF_f(D,t)$.

FSDM. Fielded Sequential Dependence Model (FSDM) [227] is another retrieval technique for Web documents [69]. The model extends SDM (Sec 2.2.6) to fields. FSDM estimates the unigram, bigram, and biterm language models from document fields respectively and interpolates their probabilities.

$$\begin{aligned} Score_{FSDM}(Q,D) = & \sum_{q_i \in Q} \lambda_U \log \sum_f P_{unigram}(q_i|D,f) + \\ & \sum_{q_{i,i+1} \in Q} \lambda_G \log \sum_f P_{bigram}(q_{i,i+1}|D,f) + \\ & \sum_{q_{i,i+1} \in Q} \lambda_T \log \sum_f P_{biterm}(q_{i,i+1}|D,f) \end{aligned} \tag{3.6}$$

Field-Based Language Models. Ogilvie et al. [150] used fields as document representations in the query likelihood model:

$$\begin{aligned} Score_{FLM}(Q,D) &= \log P(Q|D) \\ &= \sum_{q \in Q} \log \sum_f W_f P(q|D,f) \end{aligned} \tag{3.7}$$

$P(Q|D)$ is the probability of generating Q's terms by a language model induced from document D; $P(q|f,D)$ is the probability assigned to term q by a language model induced from field f in D; W_f is f's weight.

Kim and Croft [99], as us, used relevance models induced from fields. However, a significant fundamental difference with our work is that the relevance models were not used to directly score fields or the entire document as implied by the generative theory for relevance. Rather, the relevance models were used to assign a weight $W_{f,t}$ for each field f with respect to each query term $t \in Q$. These weights were then used in the query-likelihood retrieval model from Equation 3.7 instead of W_f which is the same weight for all query terms with respect to f. Specifically, $W_{f,t}$ is the normalized probability assigned to term t by a relevance language model induced from field f of top-retrieved documents; normalization is with respect to all fields. Thus, the suggested retrieval model is still based on scoring a query w.r.t. a field using the surface-level similarity between the two. In contrast, our models try to alleviate the vocabulary mismatch problem incurred by such scoring by using relevance models to score the fields. We note that using relevance-model-based field weights in our models is an interesting avenue for future work.

Neural Ranking Models with Fields. Zamani et al. [220] were also motivated by the idea that incorporating document fields can lead to more accurate document representation. They focused on representing the document in a latent vector space which can be further used by neural ranking models while our work focuses on inducing a more accurate relevance model. Their proposed NRM-F framework hierarchically aggregates field information into a final document representations as shown in Figure 3.3. They use multiple convolution layers to aggregate field

n-grams into an instance representation, and then take the average of multiple instances for a field representation, and finally concatenate multiple fields for the document representation.

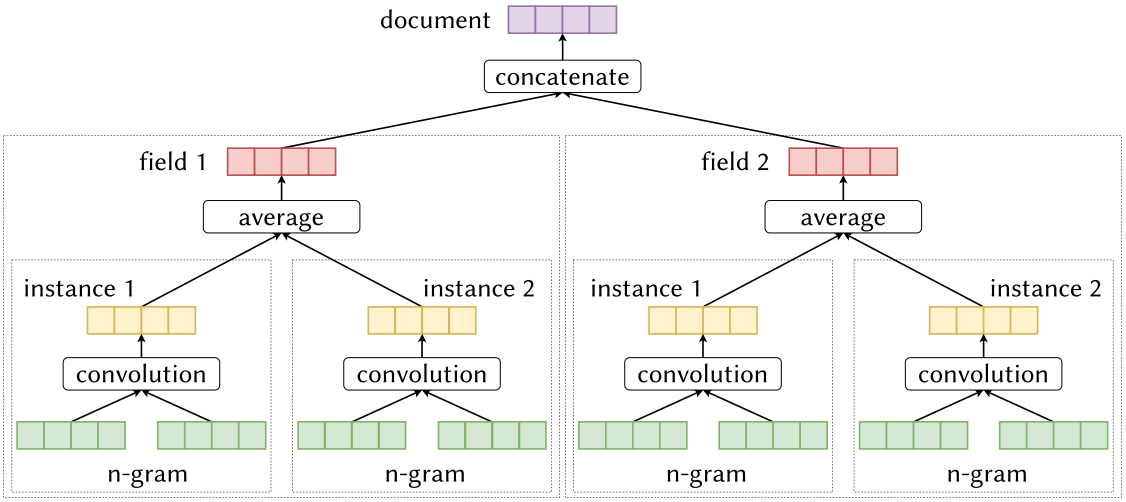


Figure 3.3: NRM-F aggregates field information for document representations.

3.3 Approach: Relevance Models with Fields

In this section, we present our methods that integrate field-based information and relevance modeling. The first method induces relevance models from fields and scores the fields independently. The second method scores fields using a relevance model induced from the entire document.

3.3.1 Field-Based Relevance Models

We extend Eq 3.4 by inducing relevance models from document fields f independently:

$$P_{RM3F}(w|f,R) = \lambda P(w|Q) + (1-\lambda) \sum_{D \in \mathcal{D}} P(w|f,D) \frac{P(Q|f,D)}{\sum_{D' \in \mathcal{D}} P(Q|f,D')} \tag{3.8}$$

where $P(w|f,D)$, the probability assigned to term w by a language model induced from field f in document D, is estimated as explained below, and $P(Q|f,D) = \prod_{w \in Q} P(w|f,D)$.

To estimate $P(w|f,D)$, we should account for the fact that fields are short, and hence, the sparsity problem is exacerbated. For example, the *title* and *heading* fields are usually much shorter than the document *body*. For ClueWeb09B collection, the average length of *title*, *heading*, and *body* are 7.22, 27.94, and 702.19 terms respectively. Thus, we use a double smoothing approach, where the maximum likelihood estimate (MLE) with respect to a field is Dirichlet smoothed with

a linear combination (Jelinek-Mercer) of field-specific and non-field-specific collection MLEs:

$$P(w|f,D) = \frac{c_{w,f,D} + \mu(\beta \frac{c_{w,f}}{|C_f|} + (1-\beta)\frac{c_w}{|C|})}{|D_f| + \mu}; \tag{3.9}$$

$c_{w,f,D}$, $c_{w,f}$ and c_w are the counts of w in field f of D, in all fields f in the corpus documents, and in all fields in the corpus; $|D_f|$ is the length of f in D; $|C_f|$ is the sum of lengths of fields f in all documents; and, $|C|$ is the number of term occurrences in the corpus; β and μ are free parameters.

To score document D with respect to query Q, we interpolate the minus cross entropy scores of applying the field-based relevance models from Eq 3.8 independently on each field:

$$Score_{RM3F}(Q,D) = \sum_f W_f \sum_{w \in Q} P_{RM3F}(w|f,R) \log P(w|f,D). \tag{3.10}$$

The field weights, W_f, are set using cross-validation.

3.3.2 Applying Relevance Models on Document Fields

The method presented above is based on scoring a field using a relevance model induced from the field. Still, fields are short and hence, the induced relevance models might not be robust. Hence, we consider a method which uses a relevance model induced from the entire document ($P_{RM3}(\cdot|R)$ from Eq 3.4) to score each of the document fields. Then, as in Eq 3.10, the scores are linearly interpolated:

$$Score_{RM3F'}(Q,D) = \sum_f W_f \sum_{w \in w} P_{RM3}(w|R) \log P(w|f,D). \tag{3.11}$$

3.4 Experiments

3.4.1 Collections and Fields

Our experiments are run on the ClueWeb09B collection which contains around 50 million English web pages. We use Indri[1] 5.12 for indexing, and apply the Krovetz stemmer to both documents and queries. Note that stopwords are removed from the query only as stopwords in the documents can have an important influence on the relevance models being induced.

We investigated three fields – *title*, *heading* and *body*. Although *inlink* is a field commonly used in other studies, we omit results for *inlink*, as our preliminary results show that including *inlink* data in the collection can have unexpected consequences. More specifically, Indri and several other systems append *inlink* data from other documents into the linked document, which can change the statistical properties of a document with many *inlink*s significantly. Our experiments

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

show that this destabilizes the relevance models being induced. We leave this unexpected finding to future work as it is an orthogonal problem to the one we wish to explore in this chapter. Note that, in our experiments, *heading* is part of *body* as it is an aggregation of the *H1*, *H2*, *H3* and *H4* HTML tags which are inside the *body* tag. We believe it might also contain useful information that can be exploited independently of the body.

3.4.2 Baselines

We used three types of existing retrieval frameworks as baselines: (1) query likelihood (**QL**) and a weighted linear combination of query likelihood over fields (**QLLF**); see Eq. 3.7 (ii) **BM25** and **BM25F**; and (iii) **SDM** and **FSDM**. For QL, the Dirichlet smoothing parameter μ is set to 2500. For QLLF, μ is 10, 100, 2500 for *title*, *heading*, and *body* and the field weights are 0.2, 0.1, 0.7 respectively. We implemented BM25F [221] (Section 4.1 of the original paper) in Indri, and followed their approach to optimize the parameter weights for {*title*, *heading*, *body*}. The implementation has been made publicly available[2]. The weights were obtained by averaging across a 5-fold cross validation. First we optimized B_f for each field independently. $K1$ was then optimized using B_f from the previous step. Finally, $W_{body} = 1$ was fixed, and W_{title} and $W_{heading}$ were swept. The final parameter choices for ClueWeb09B were $K_1 = 1.02$, *title* ($B_f = 0.36$, $W_f = 9, 2$), *body* ($B_f = 0.32$, $W_f = 2$), and *heading* ($B_f = 0.16$, $W_f = 1$). For FSDM, we used the configuration from Mohammad et al. [136]: the *title*, *heading* and *body* weights were 0.2, 0.05 and 0.75, respectively. Both ordered and unordered bigram features are only applied over the *body* field, each of which has a weight of 0.1, whereas the unigram feature of *body* has a 0.8 weight. Post-hoc spam filtering[3] is applied to all runs with a threshold of 50. Finally, we retain the top 1,000 documents for each ranked list for evaluation. Note that this would affect direct MAP or NDCG comparisons with previous TREC Web Track runs as these were scored over the top 10,000 documents.

3.4.3 Experimental Setup

An initial list was retrieved using query likelihood with Dirichlet smoothing and $\mu = 2500$. For relevance modeling, we adopted the reranking approach of Diaz [58] who showed that reranking is as effective as retrieval over the entire collection for RM3. As in Equation 3.8, $P(Q|f, D)$ was used instead of $P(Q|D)$ for relevance modeling. Thus, the document list was reranked by *title*, *heading*, and *body* query likelihood scores, and the top 50 scored fields were used for relevance modeling independently. Before reranking, we clipped the relevance models for 25 or 50 terms and re-normalized term weights. Then the 1,000 documents were reranked with RM3 over fields for the three resulting lists. Document scores from the three ranked lists were then linearly combined to produce the final ranking. A ten-fold cross validation was performed for tuning the relevance model clipping (number of terms) and RM3 query weights.

[2] https://github.com/binshengliu/bm25f
[3] https://plg.uwaterloo.ca/~gvcormac/clueweb09spam

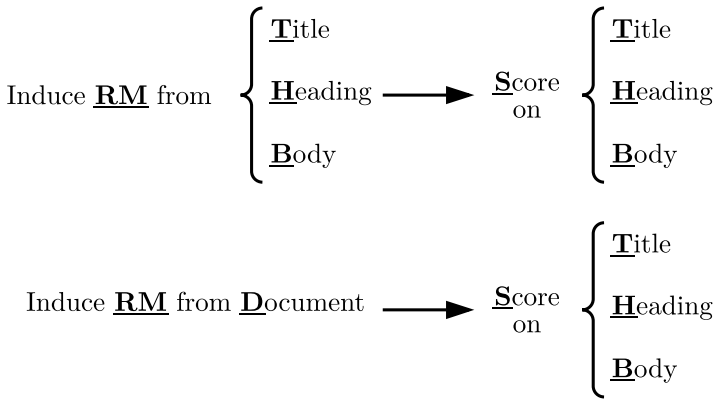


Figure 3.4: Experimental naming rules.

Table 3.2: A summary of methods developed in this work. The source column refers to where relevance models are induced. The target column refers to where relevance models are applied.

Method	Source	Target
RMFLF	All fields	All fields (weighted sum)
RMTST	Title	Title
RMHSH	Headings	Headings
RMBSB	Body	Body
RMDLF	Document	All fields (weighted sum)
RMDST	Document	Title
RMDSH	Document	Headings
RMDSB	Document	Body

Naming followed the rules outlined in Figure 3.4. All names are listed in Table 3.2. **RMTST** is a relevance model induced from the *title*s of a rank list, and used to rerank the list by scoring the *title* field. A linear combination of **RMTST**, **RMHSH**, and **RMBSB** is named as **RMFLF**. If we induce relevance models from documents, and rerank documents by *title*, we name it **RMDST**. Finally, the linear combination of scores of **RMDST**, **RMDSH**, and **RMDSB** is called **RMDLF**.

3.4.4 Experimental Results

Field-Based Models for Document Retrieval. First we consider the impact of field information on the retrieval effectiveness in Table 3.3. We can observe a consistent trend for all retrieval methods without relevance modeling: field information improves early precision. For P@5 and NDCG@20, statistically significant differences are observed, except for SDM and FSDM. Although slight AP improvements are observed in Table 3.3, the results are not significant.

Table 3.3: Effectiveness of field-based retrieval methods on ClueWeb09B. A pairwise, two-tailed t-test was performed between a non-field model (BM25, SDM, QL, and RM3) and the corresponding extended field-based model. A † denotes significance at $p \leq 0.05$.

Name	MAP	P@5	NDCG@20
BM25	0.196	0.359	0.234
BM25F	0.203	0.401†	0.256†
SDM	0.210	0.366	0.253
FSDM	0.200	0.398†	0.256
QL	0.196	0.347	0.238
QLLF	0.203	0.398†	0.256†
RM3	0.205	0.378	0.244
RMFLF	0.198	0.420†	0.257
RMDLF	0.197	0.420†	0.252

Next, we consider the relevance modeling based methods: RM3, RMFLF and RMDLF, which follow a similar trend as the methods without relevance modeling: early precision benefits from incorporating field information, but the improvements are not statistically significant for NDCG@20 or AP. The early precision of the relevance model approaches is also better than that of the baselines. In general, the results shown in Table 3.3 suggest that fields in documents can provide new relevance signals to some extent, and integrating the fields into existing retrieval models improves retrieval effectiveness; but the improvements are somewhat volatile depending on what is being measured.

Field-Based Retrieval and Relevance Modelling. Finally, we explore if the field-based retrieval methods can be further improved using relevance modeling techniques. In order to gain a better understanding, we conducted experiments using two different settings: (a) a PRF setting, where pseudo-relevance feedback documents are used; and (b) an oracle setting, in which the first five relevant (by qrel) documents from the initial list are used. As we showed in Table 3.3, using a linearly combined field-based relevance models improves early precision significantly, and we further analyze the effectiveness for each field independently now.

As described in Section 3.3, we induce relevance models using different sources: (i) the fields themselves; and (ii) the entire document. We consider the methods from the RMFLF and RMDLF family. The trends for both categories are shown under *PRF Setting* in Table 3.4. The *body* field is more effective than either *heading* or *title* fields, the *title* is slightly more effective than the *heading* field, and a linear combination of all three methods provides the greatest effectiveness improvement.

Table 3.4: Decomposition of field-based relevance modeling for both PRF and the oracle settings on ClueWeb09B. A pairwise, two-tailed t-test was performed between each model and RM3. A † denotes significance at $p \leq 0.05$.

	Field	PRF Setting				Oracle Setting			
		W_f	MAP	P@5	NDCG@20	W_f	MAP	P@5	NDCG@20
RM3	-	-	0.205	0.378	0.244	-	0.298	0.678	0.445
RMFLF	RMTST	0.1	0.122†	0.265†	0.151†	0.2	0.256†	0.646	0.432
	RMHSH	0.1	0.110†	0.246†	0.137†	0.1	0.206†	0.618†	0.393†
	RMBSB	0.8	0.189†	0.391	0.234	0.7	0.277†	0.674	0.438
	Linear	-	0.198	0.420†	0.257	-	0.293	0.732†	0.479†
RMDLF	RMDST	0.2	0.130†	0.277†	0.159†	0.1	0.152†	0.296†	0.200†
	RMDSH	0.1	0.104†	0.248†	0.136†	0.1	0.126†	0.308†	0.174†
	RMDSB	0.7	0.187†	0.380	0.231†	0.8	0.266†	0.640†	0.406†
	Linear	-	0.197	0.420†	0.252	-	0.276†	0.656	0.417†

When constructing relevance models based on the entire document and applying the constructed model to score each field (RMDLF), there are some differences from RMFLF, but the interpolated scores (Linear) perform very similar, and not significantly different.

The effectiveness of using true relevant documents to construct relevance models is shown under *Oracle Setting* in Table 3.4. This set of experiments reveals the potential effectiveness gains we might achieve using relevance modeling over fields. We observe that, if we apply a relevance model induced from the entire document to each field, system effectiveness is degraded. This confirms the observation made in the PRF experiment. More importantly, RMFLF outperforms RMDLF which suggests that inducing relevance models from fields instead of documents is a promising approach that we do not yet understand how to exploit. Experimental results of the oracle settings confirm that applying relevance modeling techniques can significantly improve the performance of field-based retrieval methods, particularly for early precision metrics.

Per-Query Performance Breakdown. In order to better understand the performance patterns, we performed a failure analysis for RMFLF on a per-query basis, as shown in Figure 3.5 (PRF setting) and Figure 3.6 (Oracle setting). In both instances we can see that the two field based methods, RMTST and RMHSH, improve performance on tail queries where RM3 and QL have low NDCG@20 scores. When considering the PRF setting and the 25% worst-performing queries for RM3, 58% and 44% can be improved by using RMTST and RMHSH, respectively. In an oracle setting, 44% and 38% of queries among the worst 25% RM3 tail queries are improved. However, none of the current field-based relevance models are robust for all queries, and the performance can be worse than either QL or non-field-based standard RM3.

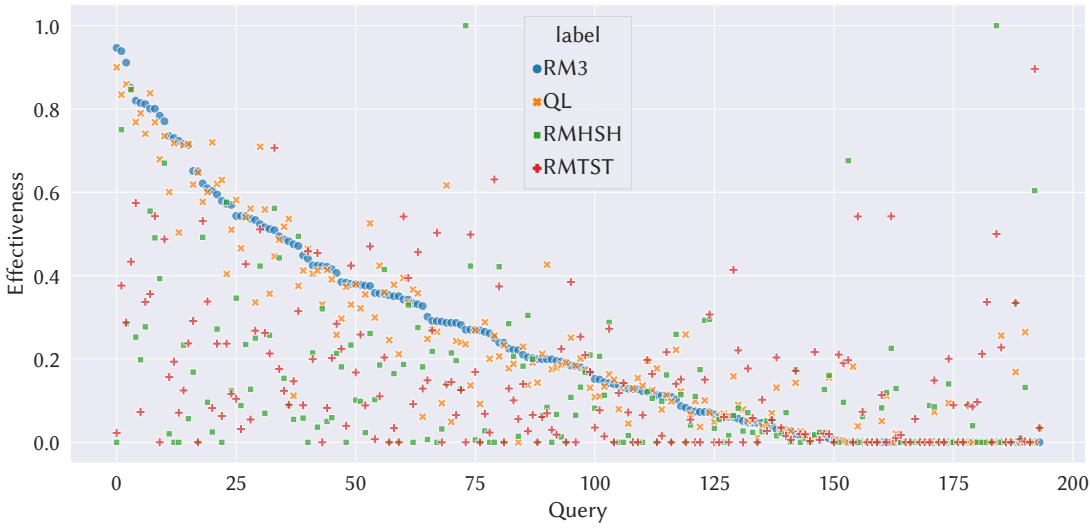


Figure 3.5: Performance breakdown for RMFLF methods on a per-query basis in an PRF setting. The evaluation metric is NDCG@20 and all topics are organized w.r.t. the RM3 methods. The "RM" means RM3, the "H" means RMHSH and the "T" means RMTST method.

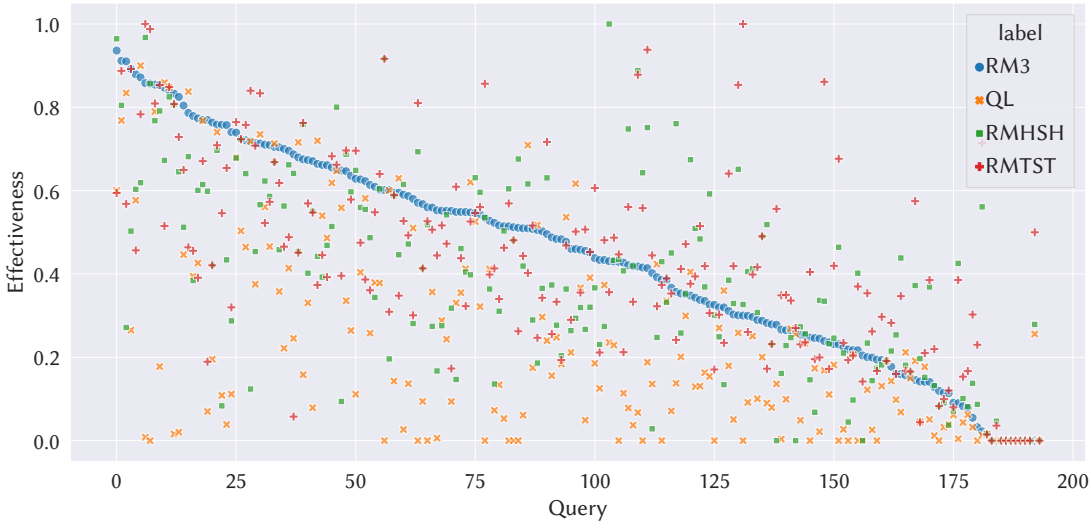


Figure 3.6: Performance breakdown for RMFLF methods on a per-query basis in an oracle setting. The evaluation metric is NDCG@20 and all topics are organized w.r.t. the RM3 methods. The "RM" means RM3, the "H" means RMHSH and the "T" means RMTST method.

Summary. We can clearly see that incorporating fields can produce modest improvements for a variety of existing models. Our field-based relevance models also make similar improvements over traditional relevance models. However, these improvements are mostly observed on early precision and are not consistently over other metrics. Can we do more? What is possible if we could *rewrite* queries to better represent the information need? In the next section we will perform a comparative analysis using rewritten queries.

3.5 Analyzing Human and Automatic Queries

The distinction between a query and the underlying information need represented by the query has been an essential component of Information Retrieval research for more than half a century. Many factors influence the effectiveness of a keyword query, and small reformulations can have a substantial impact on the performance.

Cummins et al. [49] have explored the effectiveness gap between human query formulations and automatically generated queries with a vocabulary limited to the description and narrative of TREC topics. Their experiments show that *title queries* can be largely improved by human rewritten queries. They also used a greedy algorithm to examine the upper bound of queries generated from the limited vocabulary. The optimal queries are more than twice that of the average effectiveness of human beings, highlighting the potential of query rewriting.

Recent work on user query variations have also renewed interest in this fundamental IR problem. Bailey et al. [9] have collected query variations through crowdwourding, in order to study the variability of users. Following their work, Benham and Culpepper [17] found query variations can not only provide effectiveness improvements but also improve the robustness of a search system. However, manually curated collections of queries do not necessarily translate to performance improvements in a production setting where related queries discovery must somehow be operationalized. In this section, we compare user query variations (UQV100) released by Bailey et al. [9] with automatically generated queries from Bing search logs, to identify potential gaps in automatic query generation and shed light on future work. Our findings clearly show that automatically generating queries can improve the representation of the information need. At the same time, their gaps motivate further research along the direction of query generation. Concretely, we explore the following questions: 1. Can automatically generated query variations be as effective for retrieval as carefully crafted human query formulations? 2. What are the similarities and differences between the variations being produced?

3.5.1 Comparison Method

First, query likelihood retrieval is performed using each query variation over the corresponding document collections, and then AP (average precision) is computed using trec_eval. Every query variation with a 0 AP was dropped (on average 2 queries were dropped per-topic from the human set using this methodology, and 4 from the automatic set). This is consistent with previous work

on UQVs [10, 17]. The goal of our work is to better understand how variations of similar quality that were manually generated by humans compare to automatically generated ones.

To address our first question, we gradually remove the bottom-performing $x\%$ of the Bing queries for a topic, in order to find a set of automaticly generated (selected) queries that are of comparable effectiveness quality to the human curated set. The equivalence between the two sets of queries is determined based on the median queries from the human reference sets, and the median of the current set of pruned automatic queries. A paired, two-sided t-test and a two one-sided test (TOST) can be used to identify the most appropriate cutoff threshold. TOST is commonly used in the medical community to test for statistical non-inferiority [206]. More specifically, a t-test tests for differences while a TOST test for equivalence. Once a cutoff is identified in the automatic collection which results in a similar effectiveness to the human collection, the two query sets are then exhaustively compared and contrasted.

In order to address the second research question, we explore the similarity between queries for a topic (Intra) and also compare similarity between the queries for Human and Bing (Inter). We use Jaccard similarity and *Rank-Biased Overlap* (RBO) [208] to do our intra- and inter- similarity comparisons, which is discussed more in the next section.

Document Collections and Retrieval Models. We use two test collections in our experiments: Robust04 and ClueWeb12B document collections. The Indri toolkit is used for all retrieval experiments, and stopwords were pruned from queries at runtime. Across all experiments, Krovetz stemming is applied to queries and documents, and query likelihood model [190] (Dirichlet smoothed document language model, $\mu = 2500$) was used for retrieval.

Query Variations. Instead of using title queries as originally provided by TREC, for each topic, we consider two sets of query variants. The first set were manually curated, human query variations created through crowdsourcing experiments. The second set were generated automatically using a random walk on a click graph derived from query logs in the Bing search engine. Human generated query variations for both ClueWeb12B and Robust04 are publicly available, and have been used in several recent research papers. [4][5]

The automatic query variants are generated by using a bipartite query–URL click graph taken from a 10% sample of Bing click data over several months in 2018. Note that the automatic variants are queries selected from the log as those presumably most related to the query at hand. Sheldon et al. [187] proposed a process to induce multiple query variations from a starting query or description using the random walk model originally described by Craswell and Szummer [43]. Using a two-step forward walk produces queries that would be reached if the walk starts with a single user query. Here we apply the same model, but use a two-step backward walk, which tells us what queries were the likely starting point given that we ended at the user query. The backwards walk model also performed better in the original paper [43], and produced the best

[4] https://culpepper.io/publications/robust-uqv.txt.gz
[5] http://dx.doi.org/10.4225/49/5726E597B8376

results in our preliminary tests. We did not perform additional experiments to pick the best hyper-parameters for the random walk, and leave this for future work. We note that for description queries, the query is very unlikely to occur in the graph, so temporary nodes were created for each description query that was connected to any URLs found in the description query's top-50 Bing results. Note that the descriptions used for ClueWeb12B were the backstories developed by Bailey et al. [9], and TREC descriptions were for Robust04 to ensure that the queries being generated were directly comparable to the human-generated query sets.

As a result, on average, there are around 16 automatic query variations and 12 human-generated query variations for Robust04; for ClueWeb12B, there are around 25 automatic and 39 manual query variations available. There are in total 100 topics for ClueWeb12B, and 249 topics for Robust04. For automatic variants, none were produced for three topics in Robust04, and so these were treated as empty queries in all comparisons to ensure that our results are directly comparable to previous results reported for Robust04.

3.5.2 Results and Findings

Table 3.5: Retrieval effectiveness of (fielded) relevance models on ClueWeb12B and Robust04. To find the best parameters, grid search is carried out with $docs \in \{10, 25, 50\}$, $terms \in \{10, 25, 50\}$, $orig\_weight \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Fielded relevance models are applied on *title*, *heading*, *inlink*, and *body* fields with weights 0.1, 0.1, 0.1, and 0.7 respectively. Robust04 has limited field information (e.g. only a title field "THE JOURNAL (House - April 01, 1993)") so field relevance models are not applied on it. † means $p < 0.05$ in the t-test compared to title query.

	ClueWeb12B			Robust04		
	MAP	NDCG@10	RBP@0.95	MAP	NDCG@10	RBP@0.95
Title	0.201	0.192	0.360 +0.213	0.247	0.426	0.308 +0.035
RM3 Best	0.203	0.190	0.359 +0.224	0.266†	0.429	0.321 +0.029†
FRM3 Best	0.204	0.196	0.363 +0.217	-	-	-

Retrieval Performance of Relevance Models. Table 3.5 shows the results of applying traditional relevance models and our fielded relevance models as a reference point. Significant improvements are observed on Robust04 but not on ClueWeb12B. This is in line with other work [123] that relevance models tend to work less effectively on ClueWeb12B, perhaps because ClueWeb12B is made up of web documents and is more noisy. Another difficulty of relevance models is parameter tuning. Billerbeck and Zobel [23] have shown that the best parameters of the BM25-based query expansion [171] varies wildly at the query level. This is also true for relevance models. If we were able to find the optimal parameters for every single query, on ClueWeb09B we could reach an MAP score of 0.214, and on Robust04 we could reach an MAP score of 0.284. Table 3.6 shows the statistics of the optimal choices of two parameters (the number of docu-

Table 3.6: Query-level optimal parameters of relevance models.

(a) ClueWeb09B.

		# documents		
		10	25	50
# terms	10	12	6	18
	25	6	5	13
	50	20	8	13

(b) Robust04.

		# documents		
		10	25	50
# terms	10	35	17	17
	25	25	17	13
	50	71	30	25

ments and the number of terms) at the query level. For example, 12 queries in ClueWeb12B need $\#docs = 10$ and $\#terms = 10$ for relevance models to perform optimally. The table clearly shows the diversity and difficulty in query-level parameter tuning. Importantly, even in such an optimal condition, the improvements are still small compared to what is possible with rewriting queries as shown in Table 3.7.

Table 3.7: Retrieval effectiveness of automatic query variations and human query variations. *Human* represents the median value of the human set. For Bing, median performance is reported for different filtering thresholds of bottom-performing queries. For example, *Bing 0.5* means queries in the bottom 50% are filtered out. † and ‡ mean $p < 0.05$ in the t-test and TOST test ($\Delta AP = 0.05$) compared to title query, respectively. h and b mean $p < 0.05$ in the t-test compared to human best and bing best respectively.

Query Set	ClueWeb12B			Robust04		
	MAP	NDCG@10	RBP@0.95	MAP	NDCG@10	RBP@0.95
Title	0.201	0.192	0.360 +0.213	0.247	0.426	0.308 +0.035
Human	0.178	0.190	0.351 +0.185	0.239	0.421	0.294 +0.124
Bing 0.0	0.103†	0.120†	0.230 +0.495†	0.144†	0.254†	0.179 +0.364†
Bing 0.1	0.118†	0.138†	0.247 +0.445†	0.160†	0.296†	0.204 +0.325†
Bing 0.3	0.141†	0.146†	0.284 +0.395†	0.182†	0.337†	0.226 +0.289†
Bing 0.5	**0.166**‡	**0.192**‡	**0.323 +0.313**†	0.201†	0.358†	0.248 +0.249†
Bing 0.7	0.194†	0.210	0.366 +0.271	**0.228**‡	**0.402**	**0.281 +0.216**‡
Bing 0.9	0.226†	0.243†	0.407 +0.218†	0.273†	0.466†	0.330 +0.174†
Human Best	0.286	0.304	0.501 +0.118	0.373	0.604	0.422 +0.078
Bing Best	0.239	0.252	0.428 +0.215	0.282	0.481	0.338 +0.170
Joint Best	0.288^{b}	0.303^{b}	$0.503 +0.120^{b}$	$0.389^{h,b}$	$0.621^{h,b}$	$0.436 +0.081^{h,b}$

Retrieval Performance of Query Variations. We show results for the automatic variations at different filtering cutoffs in Table 3.7. On average, both tests suggest that the (median) AP

effectiveness of automatic and human variants is statistically indistinguishable after pruning the bottom 50% queries in ClueWeb12B. We see a similar trend for Robust04 when pruning the bottom 70%. This observation is not surprising: Table 3.7 shows that the human variations for Robust04 are of higher quality than those for ClueWeb12B. Human variants for Robust04 were created by search domain experts, while those for ClueWeb12B were created through an online crowdsourcing experiment.

The most important take-away message from this comparison is that small perturbations of a query can have a significant impact on performance. For example, the original TREC title query for Bing-ROBUST is the best variant in terms of MAP for 73 of 249 queries (29%), and for Bing-CW12B, 14 of 100 queries. For human variants, it is 23 of 249, and 0 of 100 respectively. So, even in our first attempt, the original TREC title query is superior for only 1/3 of the topics. Furthermore, fusion of variants consistently outperforms a single query, even when very few variants are available [10, 17]. We do not explore this further here due to space limitations.

To answer our first question, our results show that automatic queries may be able to achieve a similar performance level to human generated ones, but a gap still exists in the percentage of low quality variants being induced through our automatic generation approach. Moreover, we can not ignore the fact that the residuals of RBP@0.95 are much larger than anticipated on both sets of queries, indicating the high level of uncertainties in our comparison, and lower than expected judgment coverage in both collections.

Topic Differences. Retrieval effectiveness varies on a per-topic basis, as do the query variations themselves, and these differences can be observed directly in Figure 3.7 and Figure 3.8, where we plot performance of pruned automatic variations and human variations relative to the median query of all known variants for that collection. We also show where the original TREC title query performance lies for that topic. As we can see, variant performance on both sets varies widely for nearly every topic. Since Figure 3.7 and Figure 3.8 suggest that there are large quality differences between query variations at the topic level, the relative differences can be further quantified using the *drop rate* shown in Figure 3.9, which is the percentage of query variations that must be pruned per-topic from the automatic set to make the two sets statistically indistinguishable.

We can see that, although on average, ClueWeb12B drops 53.8% automatic queries and Robust04 drops 61.9% to be similar to human variations, this percentage actually varies significantly per-topic: 13 topics on ClueWeb12B and 30 topics on Robust04 in the automatic set outperform the human reference set, while 12 topics on ClueWeb12B and 74 topics on Robust04 cannot achieve similar performance to the human benchmark. This is an interesting observation for several reasons. First it suggests that both approaches can produce effective queries – in fact queries that are more effective for the information need than previously known, i.e., TREC topic titles. It also suggests that despite similar overall performance, the two methods are capable of producing remarkably different query variations. This motivates us to further explore the diversity exhibited by the two sets.

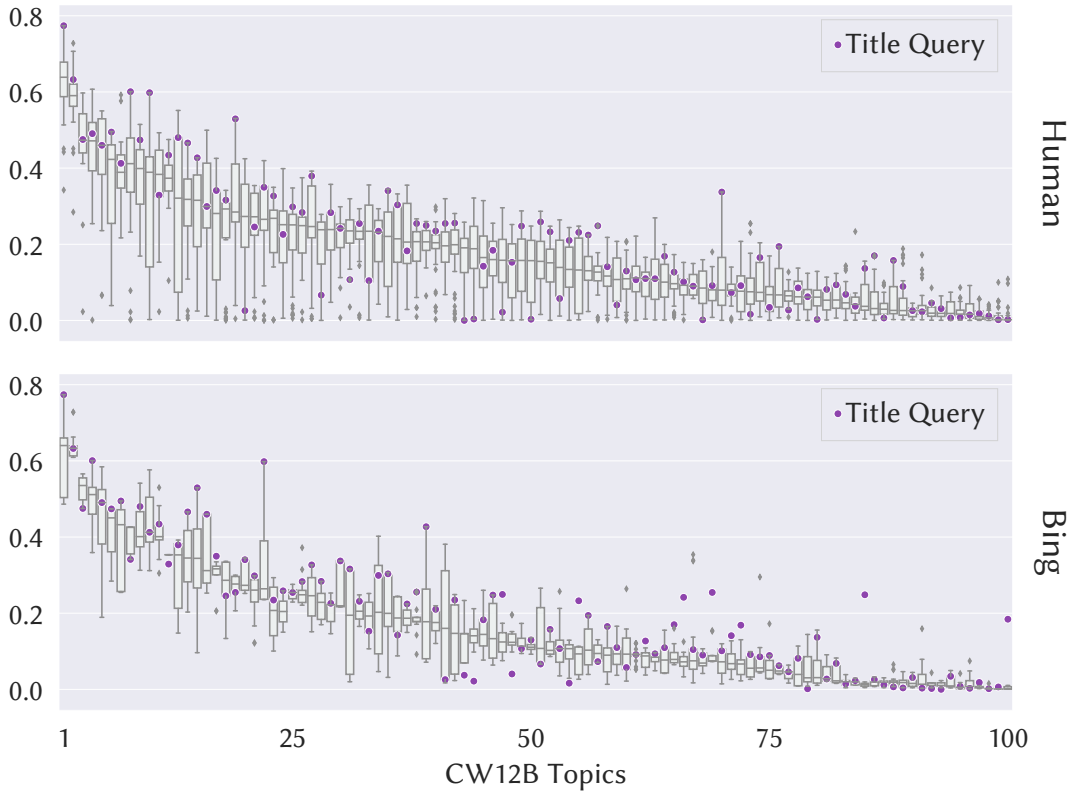


Figure 3.7: Per-topic comparisons on ClueWeb12B. Automatic query variations are in the pruned set, where the pruning percentages are 50% and 70% on ClueWeb12B and Robust04, respectively.

Term-Level Similarity. We begin to answer our second research question by exploring the similarity at the term-level. To accomplish this, we first want to determine how many queries were exactly duplicated in the human versus automatic query sets. On average, 2.7 queries match in ClueWeb12B and 0.2 in Robust04, with at most 8 and 2 matches for any one topic respectively in the two collections.

Next we extend our comparison intra-similarity: the similarity within each set of query variations. We measure the Jaccard similarity between the terms in the queries to further quantify the overlap. As shown in Table 3.8, automatic query variations on ClueWeb12B exhibit a slightly higher similarity than human variations on average per-topic — with similarities as high as 0.917 for some topics. However, Robust04 behaves differently. Here, the human generated query variations have a higher similarity. In addition, we also found that both sets of variations tend to be quite similar to TREC title queries. This is perhaps not terribly surprising as TREC title queries were formulated by topic originators, and the descriptions are simply an exposition of that intent.

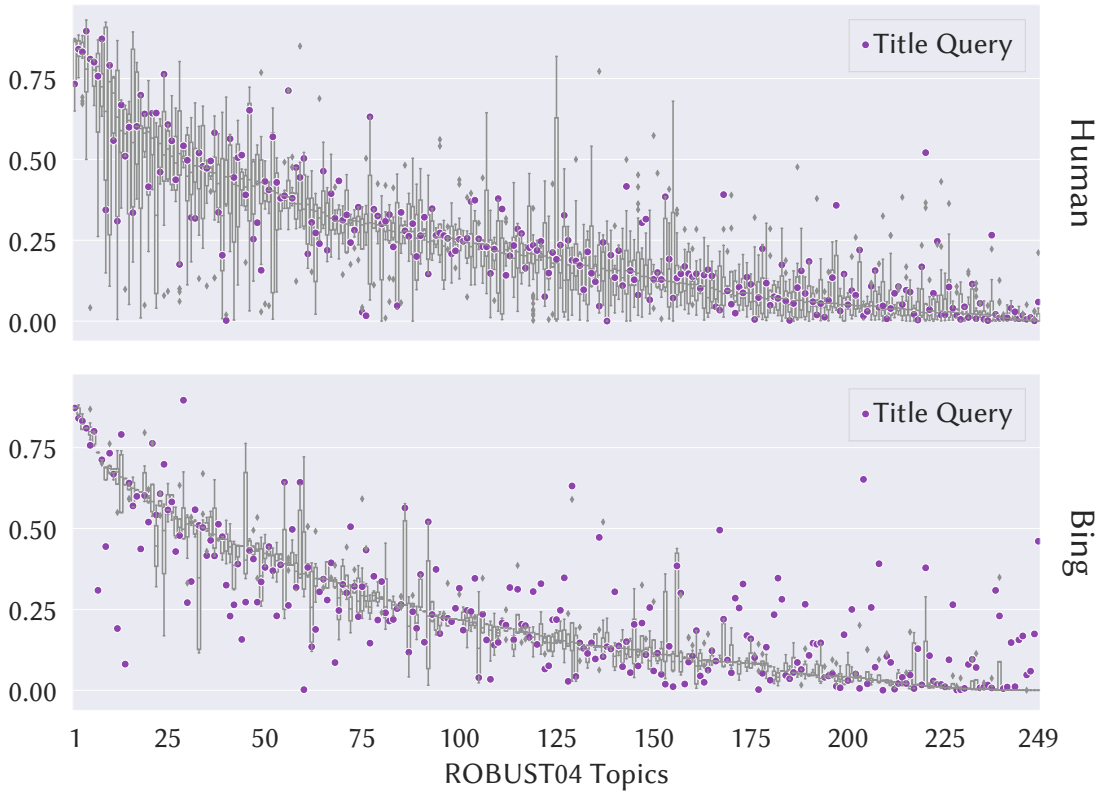


Figure 3.8: Per-topic comparisons on Robust04. Automatic query variations are in the pruned set, where the pruning percentages are 50% and 70% on ClueWeb12B and Robust04, respectively.

We also observed that automatic variations have a marginally higher similarity to title queries on ClueWeb12B than for Robust04, which we hope to explore this further in future work.

Finally, we consider inter-similarity: the similarity between the automatic and human generated queries. The average inter-similarity scores shown in Table 3.8 suggest that the two sets of queries also express the same information need in different ways, with an average similarity of 0.299 and 0.190 on ClueWeb12B and Robust04, respectively. Again, there is a difference between the two test collections: the set of human queries and automatic queries are more similar on ClueWeb12B than on Robust04, given that the maximum similarity can reach 0.971 on ClueWeb12B, but only 0.600 on Robust04. It is worth noting that the inter-similarity is lower than intra-similarity on both collections, which reinforces the inherent differences of the two sets in expressing the same information need.

Comparing Retrieval Similarities. In order to gain a better understanding of the differences between the two sets of queries, we turn now to study retrieval consistency measured with RBO,

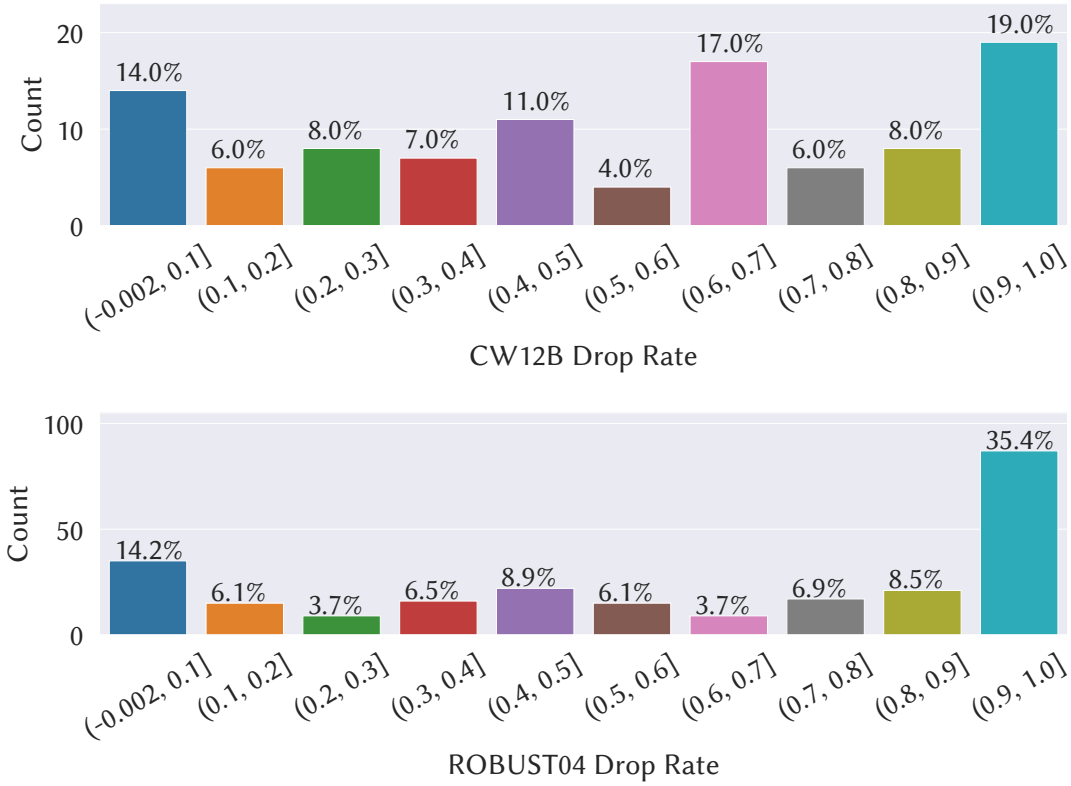


Figure 3.9: Per-topic drop rate of automatic query variations. The x-axis is the drop rate and the y-axis is the number of dropped variants.

$p = 0.9$ [208], as shown in Table 3.9, which measures similarities between two ranked retrieval lists. In general, the retrieval similarity is correlated with the term similarity in Table 3.8, except for Robust04, where automatic queries have a slightly lower level of query-level similarity but a slightly higher RBO score. There could be many reasons for this discrepancy, for example, the query length, which differs in the two collections. When considering the inter-set similarity, we observe a very low agreement between the two sets, implying that documents retrieved using the two sets of variants are in general highly diverse.

Example Queries. For some topics, human variants and Bing variants are superior in capturing different aspects of the information need, often complementing each other. We now perform a qualitative analysis for a few interesting topics on where the two sets of variants behave very differently. We found that human variants are better at addressing information needs that involve rare words or common misspellings. For Robust04 topic 301 "agoraphobia" and topic 677 "leaning tower pisa", human variants outperform Bing variants, as most human variants contain the

Table 3.8: Query jaccard similarity: within a query set (Intra), between the query sets (Inter) and with TREC's topic Title.

	Set	Intra Sim.		Inter Sim.		Sim. to Title	
		Avg	Max	Avg	Max	Avg	Max
ClueWeb12B	Bing	0.372	0.917	0.299	0.971	0.436	0.825
	Human	0.331	0.820			0.407	0.916
Robust04	Bing	0.299	0.699	0.190	0.600	0.286	0.524
	Human	0.312	0.730			0.335	0.701

Table 3.9: Retrieval consistency measured using RBO.

	Set	Intra Sim.		Inter Sim.		Sim. to Title	
		Avg	Max	Avg	Max	Avg	Max
ClueWeb12B	B	0.270	1.000	0.158	1.000	0.346	1.000
	Human	0.162	1.000			0.223	1.000
Robust04	B	0.382	1.000	0.205	1.000	0.329	1.000
	Human	0.233	1.000			0.323	1.000

correct word "agoraphobia" and "pisa", while Bing variants rarely contain the title query or contain misspelling such as "pizza". The difference stems from the generation methods for the two sets. Human variants were collected through crowdsourcing (or domain experts), during which workers were able to see the exact keyword in the information need description and then provide correctly spelled queries. Conversely, Bing variants can show a deeper understanding and capture domain knowledge as they are induced from real user queries. For example, Robust04 Topic 372 is "native american casino". Human variants were "native american casino gambling", "tribe casino gambling", and "indigenous peoples america casino gambling" which are just synonyms of the title query, while Bing variants included "indian casino", "500 nations", and "igra" which have very high specificity. Bing variants also tend to be more natural queries than human variants as crowdsourcing workers can be unconsciously influenced by the description when choosing query terms. Topic 658 is "teenage pregnancy" which also appears in many human variants, but Bing variants usually contain "teen pregnancy" and yielded significantly higher AP scores. So it would appear that both automatic and human-based approaches can be used to produce query variations effectively, but can also result in queries with very different properties. Both approaches complement each other in unexpected ways, which becomes even more apparent when considering the "Best" human, Bing, and Combined effectiveness results shown at the bottom of Table 3.7.

3.6 Summary

In this chapter, we first examined three techniques for using field information in web documents. They all outperform their non-field counterparts in terms of early precision. We then incorporated field information into relevance modeling, and observed similar trends – early precision is significantly improved. Our oracle experiments suggest that fields could be an important source of information that might be further exploited in relevance modeling. The most interesting finding is that difficult queries are improved using field-based relevance modeling.

However, relevance models are induced based on simple statistics such as term frequencies. While the technique is simple and effective, it is unable to capture more complex language properties due to the bag-of-words representations of documents. Leveraging word orders can substantially improve a ranking model's ability in deciding the relevance between a query and a document, shown by recent neural ranking methods fine-tuned on neural language models. Moreover, the queries generated by relevance models are also bag-of-words and thus are uninterpretable. These shortcomings motivate us to further explore using neural language models to capture linguistic knowledge on the input end and generate human-readable queries on the output end.

To understand the quality of automatic techniques used in commercial search engines, we also compared and contrasted automatically generated queries and human written queries for a single information need on two different commonly used test collections. Our analysis confirmed that relevance models are less effective on web collections. When they work well, the overall improvements are still not comparable to completely rewriting a query, using either automatic or human variations. We showed that while both automatically generated and human written queries can achieve comparable performance, subtle differences between the queries being created still exist. An important take-away message from our empirical analysis is that remarkable effectiveness gains are still possible based purely on the query formulation of an information need, either in the automatic and human settings. Automatic query generation is technically viable and have their own strength, but they still require a lot of efforts to achieve human level quality.

4 Generating Effective Natural Language Queries

4.1 Introduction

Effective query (re-)formulation is a fundamental research problem which has been studied extensively in Information Retrieval (IR) for several decades. While many users are surprisingly good at formulating short keyword queries to find relevant documents to satisfy their information needs, automated methods capable of generating similarly "good" queries remain an elusive research goal. In Chapter 3, we discussed relevance modeling [106] which can automatically generate/expand queries by sampling words from a language model induced from pseudo-relevant documents. But simple and powerful relevance modeling has some disadvantages that limit its performance. First, it is often implemented as a unigram model which does not take into consideration term dependencies. Useful information to understanding more complex language elements such as pronouns and co-references is lost in the modeling process. While in theory relevance models can be induced from n-grams, these approaches rarely show consistent performance improvements beyond their unigram counterparts. Second, relevance modeling is not efficient. The generated queries often contain 20, 50, or even 100 terms, depending on the *number of terms* parameter. As the number of terms grows, the time and resources required to process the queries also increase correspondingly, making relevance modeling an expensive query optimization technique. Third, since relevance models do not consider term ordering, the generated queries are not interpretable as they do not have any syntactical structure.

In this chapter, we explore using Transformers to generate queries. Before diving into the details, we would like to re-think about what a good query is. A "good" query should be readable, capture core elements of the underlying information need, and be effective (find relevant information in a target document collection – a collection which the user may have little or no information about). From a user's perspective, the query should also be short, informative, and effective. That is, the user can easily rationalize word choices and intent of their query reformulations (or suggestions) in an information seeking session in order to achieve the best out-

comes, and finding methods that are capable of mimicking this behavior can be used to improve interactions between humans and machines.

This type of "good" query, also known as a *transparent query* [138], are easy to understand, and positively influence retrieval performance for the users [101, 138, 197]. As noted by Muramatsu and Pratt [138], when a search system provides a conspicuous query rewriting process, users have clues on how best to reformulate their own queries rather than trying to guess what a system is doing surreptitiously. While previous work [138] describes how such a transformation process might be operationalized, recent transformer-based models have shown substantial improvements when applied to human text generation and query effectiveness [51]. Transformers are now readily available that contain deep linguistic capabilities (e.g. co-references) [35] which were pretrained using web-scale text collections.

In this chapter, we revisit the problem of automatic query formulation from a similar angle — document summarization. Remarkable progress has been made in the NLP community in text summarization and Natural Language Generation (NLG) in the last five years, albeit with different goals. More specifically, these new approaches generate human-readable text from documents that capture the most salient points of the target document. However, the summary may not be an effective query to find the original document or other similar but relevant documents in the collection. In IR, generating queries to find a specific document is called the *known-item finding problem* [4, 150], and can be formalized as the *strong query problem* [104]. A strong query by definition is a query that uniquely identifies a document, and was originally explored for the rank aggregation problem. Such queries are highly effective for document refinding tasks, and provide a theoretical model that can also be adapted to model effectiveness for our key task – generating effective queries for a collection. However, strong queries heavily favor rare terms and often have no conceptual overlap with the user's information need. So, a variation on this problem is the focus in this chapter. The key goal is to impose additional linguistic constraints to favor query formulations that are more human-readable.

A variation of this problem is the focus in this chapter, where we impose additional constraints that also capture important natural language properties to improve system performance. The most effective solutions for this problem are commonly biased towards rare terms in the queries generated, which are rarely representative of human generated queries, are not accurate summaries of the document, and often not easily understandable. That is, they are effective but not informative. More recently, transformer-based ranking models have become prevalent. Dai and Callan [51] and Padaki et al. [152] have shown that for these transformer-based models, long natural language queries often show higher performance than short keyword queries. This is in line with transformers' ability of capturing information from complex language. So, our goal is to generate strong queries which are readable and informative. If a generated document summary accurately captures the information need(s) satisfied by the document and is also a strong query, dramatic improvements in query suggestion / reformulation tools would be possible in the future, which have historically used user query logs and click graphs to make suggestions, rather than statistical properties of the documents relative to the entire collection [28]. While not explored

in the original work of Kumar et al. [104], the strong query problem can be easily extended to maximize effectiveness when more than one relevant document exists. In such a scenario, the model is conditioned using multiple query-document pairs.

Problem Formulation. Strong queries for a document are the shortest queries that rank a target document at the topmost position [104]. Strong queries can play an important role in understanding the performance of retrieval models, and can be used for a variety of related tasks, such as plagiarism detection. We extend this important problem by adding two additional constraints: (1) the queries should be informative; (2) the queries should not use esoteric terms (be easy to understand). We will refer to this problem the *strong natural language query* problem (SNLQ).

Let M be a retrieval model and D be a target document. Formally, the problem studied in this chapter is to construct a model to generate queries from a given document D: $D \mapsto Q$, where Q is a generated query such that: (1) $M(Q, D)$ ranks D at the topmost position (2) Q should be readable and (3) Q should also be informative to the user. Note that, our problem does not necessarily require the shortest query as there is a tension between retrieval effectiveness and quality of the summary, and short queries have the additional benefit of being more efficient to process by a search engine.

Contributions.

- ✦ We propose a novel query generation task – SNLQ. This task has the dual objective of maximizing readability, as is common in natural language processing, and retrieval effectiveness, which is often the main objective in information retrieval systems. That is, the task improves transparency when reformulating a user query but also ensures that the queries are effective when used for retrieval by the underlying search engine.
- ✦ Our solution combines Transformer-based abstractive summarization techniques and reinforcement learning over multiple objectives – summarization, informativeness, and retrieval effectiveness.
- ✦ Experiments to empirically validate the quality of our new approach using the MS-MARCO dataset show that we are able to leverage models learned from an abstractive summarizer to generate effective and readable queries for any document in the collection. We also consider how best to evaluate the quality of such queries, which can be very difficult to do in an automated way.

4.2 Related Work

There are four lines of work which are most closely related to our own: known-item finding, query generation, document summarization and interpretability of query reformulation. In this section, we discuss the connections between our work and the existing literature.

4.2.1 Known-Item Finding

Known-Item finding (or re-finding) is the task of re-finding a previously seen item. This is a common behavior exhibited by frequent users of commercial web search engines [196]. These observations motivated the work of Hauff et al. [80] who wished to explore this behavior further in a laboratory environment, which is often difficult given the lack of personalized search logs for privacy reasons. In order to construct their test collection, crowdsourcing and automatic query generation approaches were combined to simulate user behavior. The automatic generation framework used in this work was initially proposed by Azzopardi and de Rijke [4] and consisted three sampling-based methods: popular (POP), discriminative (DIS) and uniform, which simulated the process of human generated re-finding queries with "false memory". Follow-up work by Azzopardi et al. [5] applied these models on six different languages to further validate the effectiveness of these approaches. This work differs from the SNLQ problem in two ways: (1) there were no guarantees of query informativeness, and (2) their goal was not to generate a *strong query* which ranks the target document at the topmost position.

Improving search effectiveness of the known-item finding task has also been explored in past work. The most relevant to our work is the strong query problem as proposed by Kumar et al. [104]. A *strong query* is defined as the shortest query which ranks a document at the highest position. The task of strong query generation was reduced to the well-known NP-HARD set-cover problem, which motivated their greedy algorithm solution, as is common for such problems. Kumar et al. [104] concluded that a query length of less than four words is on average sufficient to induce a strong query, but no restrictions were placed on term scarcity.

4.2.2 Query Generation

The task of query generation is commonly explored from two angles: 1. given a query and the goal is to generate the alternatives; 2. given one or more documents, generate queries that are effective. The former can be used for term suggestion [84], query reformulation [88], query prediction [3] and query suggestion [25, 212]. Lee and Croft [107] explored generating queries from user-selected text that can be used to search for related information. Nogueira et al. [148] and Nogueira and Lin [146] used seq2seq models to generate queries to enrich short passages.

There are several approaches commonly used for query suggestion. For example, query log mining [25, 187] or crowdsourcing [9]. Automatic query reformulation and often rely on user interaction data and deep learning. For example, the approaches of Sordoni et al. [191] and Ahmad et al. [3] explored query reformation as a generative task for session-based retrieval, and exploited user interaction information to iteratively improve quality. The user interaction data (click information) is required by both approaches to accurately model contextual information. We approach our problem differently: (i) we do not include user interaction information, (ii) our goal is not to predict the "next query", but to generate queries for any document in a collection. Our setting, which is to generate queries from a given document, can also be linked to the large body of work on pseudo-relevance feedback (PRF) [106]. The work of Lee and Croft [107]

is closely related to our problem, where the problem explored was query generation on text passages selected by a user. They rely on phrase-based extractive summarization. Although this work is clearly related, there are a few notable differences: (1) we do not use an extractive technique; (2) we also aim to generate a natural language query, instead of only extracting a phrase directly from the original document; and (3) our problem setting also explores the importance of informativeness. Indeed, extractive summarization techniques could easily be used within our framework, but not explored here.

Query generation can also be cast as a machine translation (MT) problem [20] where a document is "translated" to "query" in order to resolve vocabulary mismatching. The "Doc2query" technique [148] uses a machine translation model to generate candidate queries based on a given document. However, instead of regarding the generated queries as new reformulations of the original user query, the queries are used to enrich the target document. This is a common technique used to learn question answering models as the overlap in terms found in the question and the answer may be minimal. Nogueira et al. [148] showed that their approach dramatically reduced vocabulary mismatches in the MS-MARCO passage retrieval task and showed that the model could be improved further in follow-on work by applying an even more sophisticated language model to the MT task Nogueira and Lin [146], Raffel et al. [164]. This work is relevant to our problem setting in two ways: (1) Our goal is also query generation from a targeted document; and (2) the generated queries should have properties similar to natural language text. Nogueira et al. [148] do not explicitly optimize for readability or informativeness, but the quality of the queries generated are consistently good when manually inspected by humans, and hence are used as a baseline in our work.

4.2.3 Abstractive Summarization

Text summarization is the process of automatically condensing natural language text into another succinct but semantically correct text representation without losing any of the key information from the original. Current NLP approaches to this problem are generally categorized as being abstractive or extractive. In this chapter, we focus primarily on abstractive summarization as this provides greater flexibility in the generated text and the learned language model. Our primary goal is to automatically discover alternative phrases for query rewriting as well as finding relevant documents that rely on alternative word choices. Recent advances in abstractive summarization models, and more generally NLG model learned with attention-based transformers are now comparable to humans (see for example [39, 61, 140]).

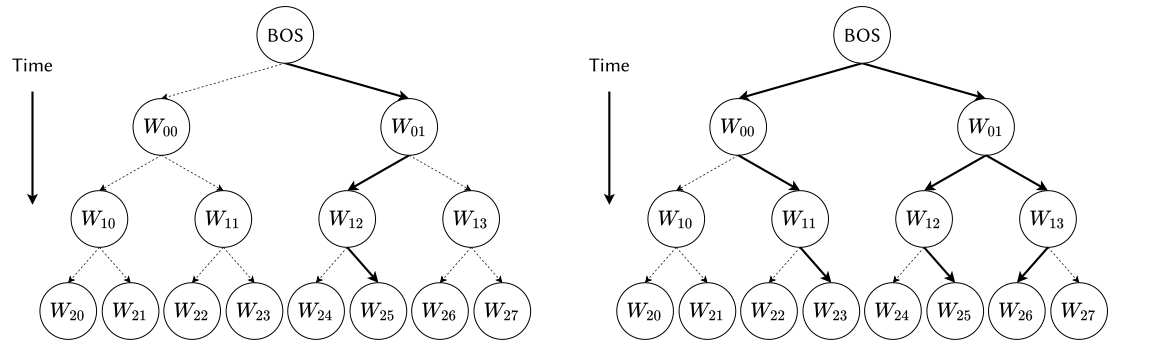
A related issue which has historically plagued NLG tasks is repeated phrase generation [157, 185]. This can now be mitigated by using pointer-generation [185] or intra-temporal attention mechanisms [157]. Our work relies heavily on these and related abstractive summarization techniques, but with an additional optimization goal – the generated summary of the document should also be an effective query that can be used to easily refind the document in the collection.

4.2.4 Interpretability of Query Reformulation

Interpretability has recently become a focus in the machine learning field, and has also been explored in IR as search engines often apply ranking models in multiple stages of the retrieval process. Several researchers have shown that improving the transparency and interpretability of a search engine can help the user word a query for their targeted information need for more effective retrieval [101, 138, 197]. Among this prior work, the "transparent query" proposed by Muramatsu and Pratt [138] is the most similar to our own. The authors show that the users prefer guidance during the query reformulation process, and a conspicuous system which provides such techniques improves retrieval effectiveness and leads to higher user satisfaction. Instead of providing explanations for each choice made by the search engine, our goal is to generate queries that are easily interpretable by a user and that also improve the quality of the retrieval results.

4.2.5 Reinforcement Learning for Seq2seq

Reinforcement learning has been widely used for seq2seq tasks to address the *exposure bias* [165] problem. Consider Figure 4.1a. When the model is being trained, it is only trained with ground truth data, while at inference time, the model needs to condition on its own output which it may have never seen during training, causing training-inference discrepancies. This problem is known as the exposure bias. Ideally, we hope to minimize training-inference discrepancies so the model generalizes well. In order to do so, we need to sample multiple generations during training and provide training signals to the model as shown in Figure 4.1b.



(a) Teacher forcing only sees the ground-truth generation trajectory, causing training-inference discrepancies.

(b) Reinforcement learning samples multiple generation trajectories and the model has been trained with more examples.

Figure 4.1: An illustration of exposure bias.

To alleviate the exposure bias problem, reinforcement learning is often used. During training, a large number of generation samples are collected and evaluated. Depending on the quality, the samples can provide positive or negative training signals to optimize the model towards producing high-performance generations.

4.3 Methodology

4.3.1 System Overview

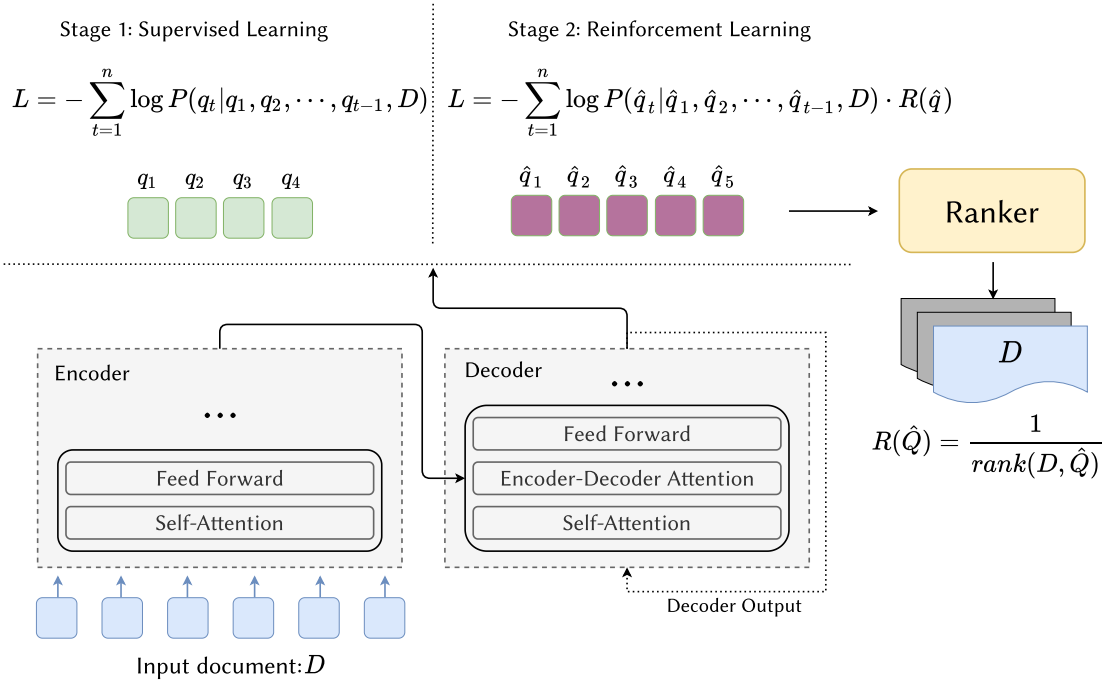


Figure 4.2: Model architecture of our proposed solution. First, documents are encoded (bottom of figure), and then are processed by a decoder generator. The figure illustrates a typical transformer in Fairseq [151], but any sequence-to-sequence model can be used. Stage 1 and stage 2 are trained on different collections, one for summarization and one for retrieval. Stage 1 training is shown in the top left pane where we take advantage of a summarization collection and maximize the quality of the text generator. Stage 2 training is shown in the top right, where the model is retrained using a document retrieval collection in order to optimize the retrieval effectiveness.

Figure 4.2 provides a high-level view of our system architecture. The only hard constraint on the summarization model deployed is that it must be retrainable directly with Reinforcement learning. We use a recent configuration of the Fairseq abstractive summarizer as described by Ott et al. [151], but any state-of-the-art model could be used, as discussed later. The pace of new abstractive summarization models being proposed makes it impossible to use the "best known" model in our experiments as it changes daily on the major leader-boards.

We have adopted a two-stage training strategy in our framework. First we fine-tune a model for the summarization task on a target collection. Once the model reliably generates high quality summaries for documents in the collection, we further train the model for our second optimiza-

tion goal: query effectiveness. The model is optimized to generate queries which rank the original documents at the topmost positions using commonly used terms whenever possible.

We achieve the effectiveness goal through reinforcement learning as it enables us to optimize non-differentiable targets directly. We cast the query generation problem as a contextual bandit problem and uses policy gradient algorithm to optimize our model. We did not use the more direct approach of training the two objectives (summarizer quality and ranking effectiveness) jointly as two unrelated objectives may not converge. More specifically, we are faced with two competing objectives which commonly rely on two different types of loss function. The readability objective generates text capturing properties from the queries and collection and is as close to the human-written ground truth as possible, while the effectiveness objective reweights term choice to improve effectiveness.

Generally speaking, text generators often rely on encoder-decoder based architectures while ranking models often rely on encoder based architectures. So combining the two objectives to find an optimal solution is a challenging problem, and is not achievable by simply combining the two objective loss functions and optimizing it combination directly.

The order of the two stages — supervised learning followed by reinforcement learning — is also important. First stage training of the summarizer ensures that the query generator generates high quality text snippets. Using reinforcement learning tasks for high-dimensional action spaces are susceptible to "the curse of dimensionality" problem [22] which can prevent convergence during training in certain circumstances [64, 218]. So, unless a properly trained summarization model is used, the query generator will select terms from a random distribution over the entire term vocabulary at each time step. Concretely, reinforcement learning is guided by the reward, which would be *zero* for randomly selected terms, and produce no gradient. This is because random terms do not capture real-world term dependencies and are not relevant to the source document or the query, and so the model gets no reward. In our experiments, we observed that the RL model did not converge until we introduced first stage training for the target collection. So, the summarizer training stage is crucial as it provides the necessary contextual knowledge between terms in a target document, and reduces the search space that the query generator must explore.

4.3.2 Summarizer

As discussed above, we use a configuration of the Fairseq abstractive summarizer as described by Ott et al. [151] as our starting point for the summarization objective. As a point of reference, the model trained achieves ROUGE-{1,2,L} scores of 41.34, 18.35, and 37.16, which are comparable with the most effective abstractive summarization results available for the CNN/Dailymail collection based on the current leaderboard for the collection. Fairseq uses a sequence-to-sequence transformer model consisting of two parts, an encoder stack which encodes the input sequence, and a decoder stack which generates sequences given the encoded information. The architecture applies attention [201], with 6 stacked layers, and each layer contains an 8-head self-attention

layer and a 2048 hidden-unit feed-forward network. We refer the readers to the Fairseq GitHub repository[1] for further implementation details.

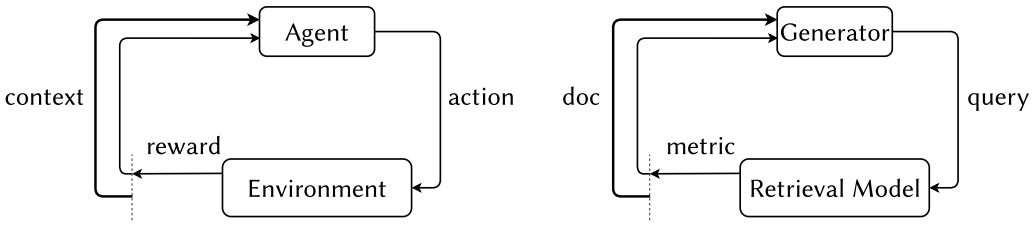
Note that we also adopt the widely used "teacher forcing" [210] method to train the model for summarization, which minimizes the maximum-likelihood loss at each decoding step. In this approach, the input token of the decoder is used as a ground truth token instead of the final token predictions at training time. Formally, given a training pair (X, Y), where X is a document, and $Y = \{y_1, y_2, \cdots, y_n\}$ is a ground-truth summary, the cross entropy loss is defined as:

$$L = -\sum_{t=1}^{n} \log\ P(y_t|y_1, y_2, \cdots, y_{t-1}, X) \tag{4.1}$$

At each time step, the output y_t is conditioned on the ground truth tokens which is different from Equation 4.2 where the actual generated sequence is used.

4.3.3 Query Generator

Once the summarization model is trained, it is retrained for the second task using the MS-MARCO collection. A key limitation of the query generation task is that there is a limited set of queries that can be regarded as "labels". However, in our task, we do know the original document used as input, and it can be used as a relevance label just as is common practice on the known-item finding task.



(a) Contextual Bandit **(b)** Query Generation.

Figure 4.3: Query generation as a contextual bandit problem.

Contextual Bandit. We model the query generation process as a contextual bandit problem. Figure 4.3 illustrates how a contextual bandit is adapted to the query generation task. In Figure 4.3a, an agent tries to find the best action given different contexts. It issues actions to the environment and receives a reward. The agent uses the reward to learn which actions are better and is optimized to favor high-reward actions. Correspondingly, in Figure 4.3b, our generator generates a query from a document, and receives an effectiveness score from the retrieval model.

[1] https://github.com/pytorch/fairseq

Given both, we are in a position where we can take advantage of optimization techniques used in contextual bandit problems, with the policy gradient algorithm being the key addition.

Policy Learning. We use a summarizer fine-tuned for the target collection as the starting point of a policy network. A policy is a decision-making rule used by the agent to determine its next action. For instance, the reward for every possible action can be estimated, and then the one with the highest reward is chosen. This is known as a *value-based method*. However, this approach is not commonly used for high-dimensional action space problems. In the query generation problem, one query can be viewed as an action, and the possible number of queries are enormous unless a set of constraints are imposed, such as query length. However, a policy-based method can be used to produce an action which it believes is promising without needing to consider the entire action space. The policy is then optimized based on the reward provided by the environment. If an action results in a high reward then it is reinforced using a policy gradient algorithm, and vice versa. For additional details on the theoretical formulation of policy gradient methods and their motivation, see Chapter 13 of Sutton and Barto [195].

Given a training document D, our training target maximizes the reward received from the generated query $\hat{Q}$, which is defined as

$$R(\hat{Q}) = \begin{cases} \frac{1}{\text{rank}(D,\hat{Q})} & \text{if D is retrieved using } \hat{Q}, \\ 0 & \text{otherwise.} \end{cases}$$

where rank is the rank position of document D in a rank list scored with a retrieval model M. The reward can also be defined to incorporate other features such as query length. In our model, we use the REINFORCE [209] training algorithm, and the loss is defined as:

$$L = -\sum_{t=1}^{n} \log\ P(\hat{q}_t | \hat{q}_1, \hat{q}_2, \cdots, \hat{q}_{t-1}, D) \cdot R(\hat{Q}) \tag{4.2}$$

Note $\hat{q}_t$ is conditioned on the actual output sequence $\hat{q}_1, \cdots, \hat{q}_{t-1}$ and ground truth labels $q_1, \cdots, q_{t-1}$ are not needed. This loss is intuitively interpreted as follows: the greater the reward, the more likely the model will generate similar outputs. For example, if the probability of choosing a token is 0.5 and it delivers a very high reward, the probability of choosing this token again in the future will be increased as training continues.

4.4 Baselines

We compare our proposed method against a few baselines. All the approaches used for our empirical evaluation are described in Table 4.1. The subscript p is appended when query length is sampled from a *poisson* distribution, and f is used when the query length is *fixed*, which will be explained shortly. Two additional points of reference are provided that represent the two best models currently available for the abstractive summarization task for the CNN/DailyMail collec-

Table 4.1: A summary of all methods used in this work.

Method	Description
POP	Generate queries by popularity sampling.
DIS	Generate queries by discriminativity sampling.
GREEDY	Generated queries by greedy algorithm.
PSG	Use passage beginning as query, truncated to match queries generated by our method.
T5	A high-quality abstractive summarizer proposed by Raffel et al. [164] and fine-tuned using query-document pairs as described by Nogueira and Lin [146].
UniLM	A high-quality abstractive summarizer recently proposed by Dong et al. [61]. As with T5 [146], we fine-tuned the model using query-document pairs for query generation.
RL	The reinforcement learning model proposed in this chapter.

tion – UniLM and T5– both of which consistently generate high-quality natural language text. In our experiments, we also fine-tune UniLM using query-document pairs and T5 in the same way as initially described by Nogueira and Lin [146]. Both can be regarded as reference points for query quality, and are much larger (and recent) pretrained transformers than the Fairseq summarizer we had available when initially undertaking this work. However, as demonstrated in Table 4.2, high quality summaries for a document are not necessarily effective queries. The generated summaries from these two algorithms were truncated as this makes them more comparable to our methods while still retaining the quality generated text.

Popularity Sampling (POP). The popularity sampling baseline was initially proposed by Azzopardi et al. [5], and favors popular or frequent terms in a document. Formally, terms are sampled from a mixture distribution of the document and the collection.

$$p(t|\theta_d) = (1-\lambda)p(t|d) + \lambda p(t) \tag{4.3}$$

where

$$p(t|d) = \frac{n(t,d)}{\sum_{t'} n(t',d)}, \quad p(t) = \frac{n(t)}{\sum_{t'} n(t'i)} \tag{4.4}$$

where $n(t,d)$ is term occurrences of t in d, and $n(t)$ is term occurrences in the collection.

Discriminativity Sampling (DIS). Discriminativity sampling is our second baseline, and also from Azzopardi et al. [5]. This method favors terms that are rare. It is proportional to the inverse

term collection frequency. Compared to POP, the difference is how $p(t|d)$ is defined.

$$p(t|d) = \frac{b(t,d)}{p(t) \cdot \sum_{t' \in d} \frac{b(t',d)}{p(t')}} \tag{4.5}$$

where $b(t|d) = 1$ if t is present in d. For all terms in a document, the only variant is $p(t)$, so the probability of a term being sampled is proportional to the inverse term collection frequency.

Greedy (GREEDY). This is the original strong query algorithm proposed by Kumar et al. [104], which generates the shortest query capable of ranking a document at the topmost position. The algorithm incrementally selects the rarest term from the document, until the intersection of the documents containing these terms contains only the target document. Refer to the original paper for further details [104].

Note that this algorithm has an important limitation on large collections. The algorithm behaves erratically when the target document is a subset of other documents, or if there are duplicates in the test collection. So, in some cases, the algorithm will not terminate until every term in the document has been added to the query. In the MS-MARCO collection, we encountered several instances of this erratic behavior as there are several (near-) duplicate documents. In such cases, we truncate the queries to 5 tokens. During our experiments, we found this bound to be more than sufficient to find the original document very effectively in the vast majority of cases.

Passage (PSG). We also use the corresponding passage of a document released in the collection as a reference query. While long, these are human-readable and often highly effective as they are extracted directly from the original document. Human crafted summaries would perhaps be a more interesting baseline and exhibit higher variance, but are not available for MS-MARCO at this time.

Since the passages are extracted from the target documents, it is not surprising they are highly effective. We also extract the first a few tokens from the passages to match the query lengths of other methods for the fairest comparison. This is similar to the commonly used NLP approach where the first few sentences are used as a summary baseline, and is often difficult to beat in practice [185], particularly for news documents. When the query length is sampled from a *poisson* distribution, we refer to it as PSG_P; when the length is same as the query generated by our network, we refer to it as PSG_F.

Neural Summarizers. T5 [164] and UniLM [61] are the state-of-the-art techniques in abstractive summarization. They are both versatile seq2seq models which were trained by utilizing transfer learning with several text-related tasks. Nogueira and Lin [146] have successfully used a T5 model to enrich sparse documents. While these neural summarizers are surprisingly good at generating high quality summaries, it is unclear if the quality can be directly translated into retrieval effectiveness for ranking tasks. We fine-tuned T5 and UniLM using relevant passage-document pairs as Nogueira and Lin [146].

4.5 Experiments

4.5.1 Collection Details

We use the CNN/Dailymail collection to train and validate the summarizer, and MS-MARCO to train and test the query generator.

CNN/Dailymail. CNN/Dailymail is a large news corpus commonly used for the summarization task. Every document in the dataset contains an article and a few human-written highlights, which can be used as the ground truth. We truncated documents and highlights for training efficiency as in prior work [140, 157, 185]. Specifically, source documents are truncated to 800 terms and the highlights to 100 terms. The collection is split into train, valid, test splits based on publicly available data.[2] The final dataset contains 287,227 training pairs, 13,368 validation pairs, and 11,490 testing pairs.

MS-MARCO. The MS-MARCO collection contains both document ranking and passage re-ranking tasks, with 8.8 million passages extracted from 3.6 million web documents. The document ranking and passage re-ranking tasks share the same queries. In our experiments, we removed around 3.5% of the documents which contained serious parsing errors, or that contained fewer than 25 terms, as these contained uninformative content such as "this site requires cookies to be enabled to function". The qrel files released with document ranking task and passage re-ranking task were joined to map a total of 309,387 document-passage pairs, and used for training and testing in our experiments.

Our validation set contains all the unique documents extracted from the document validation qrel file.[3] Training and testing sets were constructed as follows. All unique documents in the document training qrel file [4] minus documents in the validation set were randomly sampled to produce test and training. The final split contains 299,535 training examples, 4,926 validation examples, and 4,926 testing examples.

Shared Vocabulary. In order to transfer the model from one collection to another, a joint vocabulary was created. We used the spaCy[5] toolkit to tokenize and process the text and applied subword-nmt[6] as is common practice. A vocabulary size of 40K was used.

Query Generator Parameters. Queries were generated using beam search with a beam size of 5. Query lengths are determined in two ways, fixed or sampled. For fixed, the target was 10 tokens per query. Note that our summarizer was also initially trained with a 10 term target, as

[2] https://github.com/abisee/cnn-dailymail
[3] https://msmarco.blob.core.windows.net/msmarcoranking/msmarco-docdev-qrels.tsv.gz
[4] https://msmarco.blob.core.windows.net/msmarcoranking/msmarco-doctrain-qrels.tsv.gz
[5] https://spacy.io
[6] https://github.com/rsennrich/subword-nmt

user queries rarely exceed this length [42]. We also sample lengths from a *poisson* distribution between length 3 to 10 for comparison purposes.

4.5.2 Automatic Evaluation

To evaluate the effectiveness of the generated queries, we use reciprocal rank and average rank position. As millions of iterations are required during the reinforcement learning stage, efficient ranking is critical. So, we used the Pisa search engine[7] and BM25 to iteratively rank documents and evaluate the reciprocal rank of the documents during the learning phase.

How best to evaluate fluency or readability of the generated text is less straightforward as there is no standard benchmark unless human judgments are performed. The general best-practice is still to have both human and automatic evaluation results when assessing natural language text [79]. So, we have considered many measures in order to capture different dimensions of quality, including crowdsourcing of human assessments to ensure the validity of our conclusions. We will refer to automated methods that measure some notion of natural language text quality using the generic term READ_A. Human evaluation is discussed further in the next subsection.

The first READ_A measure we consider is the Flesch reading-ease test. It is calculated based on the # of syllables, # of words, and # of sentences. Unfortunately, this metric does not account for word order. We also considered ROUGE where the original document is the reference. While researchers are often skeptical about ROUGE, we find that it does indeed correlate with our human assessment as it can capture basic higher order dependency information, but not informativeness. The only other recent solutions found in the literature leverage Perplexity [93]. The measures proposed by Kann et al. [93] are somewhat intuitive in that the idea is to get an average perplexity of the terms in the summary. However, no publicly available implementation of SLOR or WP-SLOR is currently available, and so we have computed perplexity using the pretrained GPT model[8] which is known to generate high-quality text, and normalized by the length of the query. We refer to this automatic method as PPL/QL henceforth.

4.5.3 Human Evaluation

As is common with all abstractive summarization techniques, the only way to confidently assess the quality of the generated text is to perform human assessments to judge the quality. So, we validate our findings using a human assessment exercise.

We gathered judgments for both Readability and Informativeness, which is standard practice when assessing the quality of natural language text generators [66]. Readability is synonymous with the grammatical correctness of the text [38, 198] and informativeness refers to the "meaning" [198] or "importance" [38]. Readability measures if the query is grammatically correct and

[7] https://github.com/pisa-engine/pisa
[8] https://github.com/huggingface/transformers

fluent; informativeness measures if the query is informative and representative with respect to the document. In contrast to READ_A, we will use READINF_H to refer to the human judgments.

Method. We performed human evaluation using a crowdsourcing exercise on Amazon Mechanical Turk.[9] We cast the evaluation process as a headline quality evaluation which is a well-defined task in the NLP community. We gathered judgments for 50 documents sampled from our holdout set. The text of document was shown to an assessor, and informed them that they would be judging a series of headlines that are automatically generated for each. Then we asked the assessor to evaluate the quality based of two different criteria: readability and informativeness. Assessors were asked to evaluate each headline using a 5-point scale. The criteria used for assessments was:

- **Readability** – is the headline grammatically correct and fluent, where a 0 = "major errors, words have no apparent ordering" through 4 = "Outstanding, grammatically correct English sentence".
- **Informativeness** – is the headline informative and representative, where 0 = "incoherent and completely unrelated" to 4 = "Complete, the most important meaning is preserved, and is easy to understand".

For each document, we create two tasks, one for the *poisson* set and one for the *fixed* set so that they were compared directly and independently with the most appropriate counterpart. So one task presented GREEDY, PSG_P, POP_P, DIS_P and RL_P queries, and the other task presented GREEDY, PSG_F, POP_F, DIS_F and RL_F queries. The queries were shown in random order to keep the assessors from inferring any pattern. In addition, a third task was run for the same documents to gather assessments for the current state-of-the-art methods: T5 and UniLM. Both of these models are capable of generating high quality natural language text and provide an important point-of-reference for future work.

A screenshot of the user interface is show in Figure 4.4. Note that five headlines were shown at one time, but are trimmed to two in the example image.

Crowdsourcing Aggregation. We collected five assessments for every generated query in the 50 document set, and several pilots were run to determine the most suitable configuration for the crowdsourcing study. Assessors were paid $20c USD per hit, with worker eligibility requirements set to Master, greater than 10,000 approved HITs, and the Turker had to have a HIT approval rate for all HITs greater than 97%. A total of 32 assessors participated in our study. 9 assessors completed only 1 assignment. 7 assessors completed 2-9 assignments. 16 assessors completed at least 10 assignments. The mean and median of tasks completed by one assessor are 20 and 10 respectively. On average, each assessor spent 150 seconds to complete 1 assignment. In order to validate the quality of our judgments, we computed the inter-assessor agreement using Krippendorff's α [102], which was 0.552 for readability judgments and 0.472 for informativeness judgments.

[9] https://www.mturk.com

This is in line with similar crowdsourcing experiments which incorporate graded relevance, and in order to further improve the reliability, we applied the EM (expectation maximization) algorithm [55] on the 5 judgments per query we collected. In order to generate the final scores for both criteria across all methods. This approach has been shown to reliably normalize judgments and make them more resilient against assessor disagreement problems in crowdsourcing experiments, particularly when graded assessments are used [163], and has been used in the same manner in several other recent studies [86, 90, 91]. We used the publicly available EM implementation of Sinha et al. [189] to create the final labels.

We also used a simple quality control test during the exercise to filter out assessors who were not clearly not assessing correctly. For this, we compared the informativeness score given for the original passage snippet against their score for the greedy algorithm, which is often one or two terms and should be the least informative. If they scored greedy higher than the actual passage, their judgments were dropped, and they were blocked from participating in the exercise.

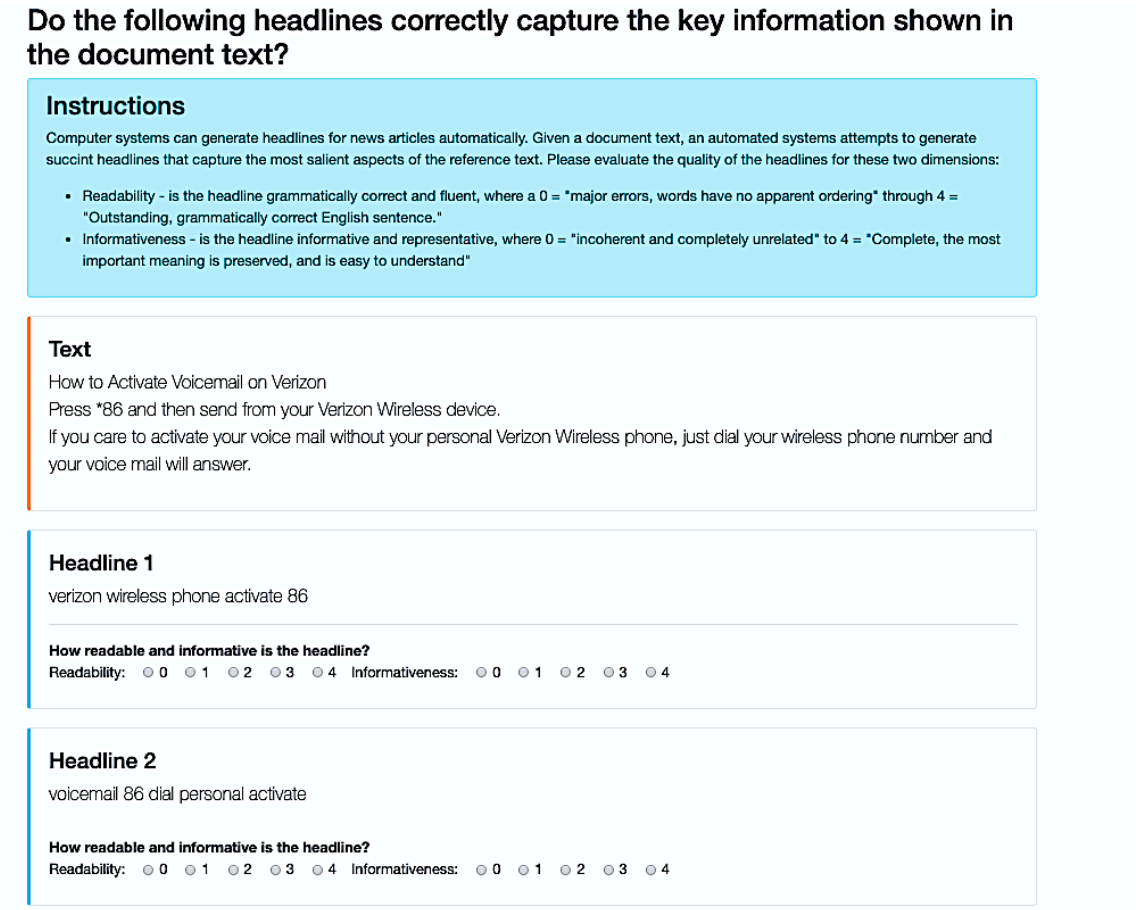


Figure 4.4: Crowdsourcing interface for readability and informativeness assessments.

Document Selection. The MS-MARCO collection contains a substantial number of parsing errors and uninformative pages. For example, on a web page, "Q&A_Children can suffer" was incorrectly parsed as "Q&AChildren can suffer" (the space between "A" and "Children" is missing) and ends up with a rare but invalid word "achildren" which can be easily exploited by the GREEDY and DIS algorithm. Another example is a web page consisting of many hex color codes was excluded for evaluation as the queries produced by all the methods were effective but also were not informative as they did not align with the original reference queries. While walking the list of available documents, we attempted to select 50 documents of reasonable length and that were parsed cleanly by the creators of the original MS-MARCO collection.

4.6 Results

Table 4.2: Retrieval effectiveness and readability/fluency comparison for all 12 methods. Our approaches are shown in bold face.

Model	Len	RR	Rank	Scarcity	PPL/QL	ROUGE Precision				Read
					Avg	1	2	L	W	Flesch
PSG	55.1	0.790	67.3	2.9%	7.7	91.31	81.67	87.88	77.72	21.93
GREEDY	2.4	0.923	2.3	53.1%	51 260.7	100.00	0.19	85.85	80.01	4.96
UniLM	5.6	0.261	146.5	3.6%	113 412.1	82.84	40.00	81.48	65.65	71.94
T5	5.9	0.232	200.0	3.1%	11 501.3	80.73	35.64	79.02	62.77	75.64
PSG_P	5.8	0.347	239.2	0.5%	8282.6	93.68	83.25	93.36	81.36	67.16
POP_P	5.8	0.246	414.6	2.0%	28 466.0	86.23	4.43	74.03	56.53	71.92
DIS_P	5.8	0.847	11.8	52.6%	17 949.3	73.11	0.73	54.84	42.00	37.22
RL_P	5.8	**0.530**	**114.5**	1.1%	22 297.5	94.04	18.77	80.35	62.54	69.00
PSG_F	9.6	0.519	102.7	0.7%	307.6	93.76	84.18	92.82	80.22	66.43
POP_F	9.6	0.492	159.5	3.5%	2586.8	86.02	4.38	67.59	46.78	68.64
DIS_F	9.6	0.886	5.7	54.1%	1878.9	66.65	0.78	46.02	31.76	37.82
RL_F	9.6	**0.780**	**11.0**	1.0%	1128.1	94.88	20.11	75.34	53.68	68.98

4.6.1 Retrieval Effectiveness

The retrieval effectiveness results are shown in Table 4.2. First, we consider the impact of query length. PSG and GREEDY queries represent both ends of the spectrum of query length. The former is formulated using natural language and is verbose. In our experiments the average length of PSG is 55.09 terms. Conversely, GREEDY extracts the most discriminative terms yielding queries that contain 2.44 terms on average, which is even lower than the average of 3.26 originally reported by Kumar et al. [104].

We report both reciprocal rank (RR) and average rank for effectiveness comparison as RR is the evaluation method used to determine the leader-board for MS-MARCO. Both PSG and GREEDY queries are highly effective. The reciprocal rank of PSG is 0.790 and an average rank is 67.3. Note that we use a simple bag-of-words ranker (BM25) and performance would very likely be improved using any of the more complex deep learning models that dominate the leader-board. Nevertheless, the queries induced are highly effective with even simple ranking models. So, while often effective, long passages extracted directly from a document are not necessarily good queries for refinding the document. Given that there are similar or even duplicate documents in the collection, the variance is as expected. The GREEDY algorithm is designed specifically to retrieve the original document, and this is confirmed by the impressive RR and average rank effectiveness the model achieves. However, duplicate documents again have an impact on the quality.

Following GREEDY in Table 4.2 are the two best abstractive summarizers T5 and UniLM. They achieve surprisingly low retrieval effectiveness, further justifying the use of a reinforcement learning retraining stage as proposed in this work. We suspect that their low effectiveness may be related to their "abstractive" nature which can improve the linguistic content, but not refinding.

We now turn to the Poisson sampling experiments. For each document, we sample a length from the Poisson distribution, and all systems generate queries of that length. In this group, PSG_P and POP_P queries have the worst reciprocal rank and average rank. This is understandable as PSG_P can contain several stop words. The same limitation affects POP_P as sampling is based purely on frequency, and stop words have a high probability. DIS_P performs the best among this group, with a reciprocal rank of 0.847 and average rank of 11.8. Essentially, DIS is a non-deterministic variation of the GREEDY algorithm, resulting is a similar behavior of choosing rare words as a query. Both methods exploit rare terms in a document, so very few terms can often be found to reliably refind the original document. Our proposed RL model achieves a reciprocal rank 0.530 and average rank 114.5, and as we will show shortly also performs well for our other target goal – informativeness / readability. When considering only effectiveness, it falls between POP_P and DIS_P, and is consistently better than PSG_P.

The final grouping compares fixed length generation. While 10 terms is rather long for a "normal human query", it is a realistic target for a compressed summary that retains some natural language properties, thus making them more understandable (hopefully). Compared to the Poisson results, we see that retrieval performance improves as the query length grows. This is again not unexpected as verbose queries often favor precision over recall [77]. Note that the PSG_F and POP_F queries show considerable improvements in this configuration when compared with DIS_F queries. This is because PSG_F and POP_F queries are more likely to incorporate at least one or more discriminative terms as the length increases. Another major improvement can be observed for RL_F. This could be an artifact of the initial summary training stage which also targeted ten terms, and will be investigated further in future work.

In general, our retrieval results align with our intuition. PSG queries are extracted from the original document and do not suffer from vocabulary mismatches, so they are often effective queries for refinding. POP is inferior to DIS. With very few rare terms, GREEDY can consistently

produce short and effective queries. Our RL method also reliably improves the effectiveness of the queries, easily outperforming PSG, POP, UniLM and T5, and is nearly as effective as DIS when query length is increased.

4.6.2 Automatic Evaluation Results

As shown in the previous section, retrieving the original document is not a difficult problem from an algorithmic perspective if only rare terms are used. But how readable and informative are such queries? Do they capture the most salient aspects of the underlying document? Can users easily understand their intention? The answer is almost certainly no in many cases. So there is a natural tension between the effectiveness of a query and the interpretability of it.

Recall in Section 4.5.2 that reliably measuring fluency in an automated manner is still an open research problem. In Table 4.2 we compare a variety of different evaluation metrics in order get a better indication of the quality of the queries being generated. All of these metrics have been used for automated evaluation of human generated text in the past. The first metric we consider and arguably still the most commonly used one for summarization tasks is ROUGE. We report only precision-based ROUGE where the document is the reference. PSG's ROUGE scores are high, but not 100 as expected because there are a few incorrect document-to-passage mappings in the MS-MARCO collection. The creators have acknowledged that the passage corpora and document corpora are collected at different times, causing such inconsistencies. Overall, queries derived from passages all have high ROUGE scores. For GREEDY, ROUGE-1 clearly shows that all the terms are from the document. ROUGE-2, ROUGE-L and ROUGE-W may not be meaningful for the GREEDY algorithm as 2,414 out of 4,926 queries contain only one term, resulting in a score of 0 for ROUGE-2 and score of 100 for ROUGE-L and ROUGE-W.

The ROUGE scores of UniLM and T5 also have several notable properties. Both systems prefer more bigrams (high ROUGE-2) than the other methods, and maintain term ordering (high ROUGE-L) even when they are not adjacent. This is an important contributing factor to their highly readable text.

The ROUGE scores for the Poisson sampled methods reveal that DIS_P are the least compliant for ordered sequence measures, and POP_P also commonly permutes term orderings. The RL method achieves higher scores especially in ROUGE-2, ROUGE-L, and ROUGE-W, which is promising as it suggests the model has retained its NLG capabilities learned during summarization. Similar trends can be observed in the fixed length grouping.

The last automatic metric we consider is the Flesch score. The readability score for PSG is low since punctuation was removed for query purposes, thus removing the sentence normalization factor. GREEDY queries also perform quite poorly. Despite being short, the number of syllables in the terms selected are large, indicating they are indeed rarely seen terms, such as "nebuchadnezzar" or "hypochlorous". For all other methods, Flesch scores are similar except for the DIS which also favors rare terms.

While ROUGE and Flesch provide some insight into the characteristics of these queries, they capture little or nothing about grammatically, informativeness, or fluency. So, our final automated

evaluation technique leverages perplexity, which we derive using a high quality pretrained language model OpenAI GPT. The score produced is inversely proportional to the chance of a query being a proper sentence in the collection using a high quality NLG model. PSG has the lowest score, or highest chance of being natural language. GREEDY has the highest perplexity score except for UniLM which we will analyze later, again illustrating the two ends of the spectrum. For Poisson sampled methods, PSG_P has the lowest score, POP_P is least likely based on the reference distribution, and surprisingly DIS_P is the best, with our RL method falling somewhere in the middle. For fixed length methods, DIS_F is still better than POP_F, which is also surprising. For longer queries, RL outperforms both POP_F and DIS_F.

Effectiveness-Readability Tradeoffs. Until now, we have only considered effectiveness and quality as independent goals, but recall that our real goal is to find the best trade-off between both of these objectives. This is best visualized by the Pareto frontier between readability and effectiveness. Figure 4.5 shows the trade-off between retrieval effectiveness (reciprocal rank) and query fluency measured as the perplexity normalized by query length. The best performing methods appear in the top left quadrant of the tradeoff graph. We can see that our proposed RL approach does indeed find the best balance between the two competing goals.

Since the retrieval problem is trivial for DIS_P and GREEDY if the rarest terms can be selected from a document (even parsing errors), we also explore their performance when the vocabulary is limited. We explored limiting the vocabulary for the two methods to the most frequent 50K, 100K, 200K, and 500K terms, and measured the performance impact. Figure 4.6 shows the changes in performance for both of these algorithms when allowed to only use the most common 50, 100, 200, or 500 thousand terms in the collection. Overall DIS_P is highly sensitive to vocabulary size. We also observed that the performance of DIS_P can be unstable when using a limited vocabulary. The reason is that rare terms are no longer available, and so term impact varies less, and random sampling increases the variance as the term frequency distributions become less skewed. The GREEDY method is remarkably consistent across different vocabulary sizes, as it can always take full advantage of the rarest terms remaining in any vocabulary.

Results for T5 and UniLM can be seen in the bottom right corner of Figure 4.5. We observed high perplexity for these models. This is somewhat unexpected, but we believe it is related to how we have conditioned perplexity on the GPT language model, and not the models used to generate the queries. Care should be taken when using perplexity to automatically evaluate quality, as we have observed that the results are strongly dependent on the text the language model is conditioned on.

4.6.3 Human Evaluation Results

Given the newly created judgments gathered with Turk we can properly assess the quality of our results in terms of human readability. The key results are shown in Table 4.3. Similar to the complete holdout test set results, we also computed a battery of the automated metrics. The automated metrics do indeed demonstrate similar trends in Section 4.6.2 when compared directly

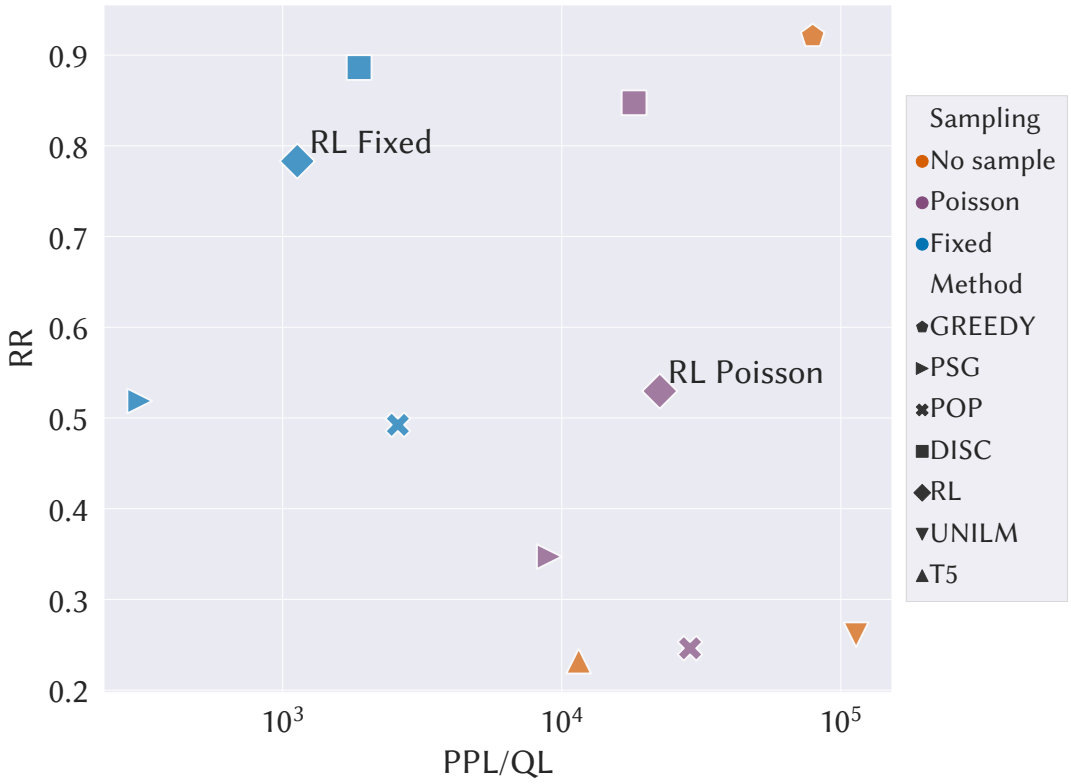


Figure 4.5: Effectiveness-readability tradeoffs according to automatic metrics on the holdout set. Lower perplexity is better, so a system located in the top-left has the best trade-off.

against the human assessments. The final human assessed scores are shown on the right of the table under "Human/Readability" and 'Human/Informativeness". The PSG_P and PSG_F tend to result in high quality queries as they were extracted directly from the documents, but truncation clearly has an effect on both. The longer the query, the better they perform. Interestingly, UniLM and T5 received the highest readability and informativeness scores after PSG_F, indicating they are very good at representing the document and they can generate coherent and succinct queries that are of high quality. However, as discussed previously, they are not necessarily effective queries. When compared to other baselines, our approach RL_P and RL_F are significantly different based on our preliminary human assessments.

Tradeoffs Revisited. We are now in a position to fully assess our primary objective, which was to find the best tradeoffs between effectiveness and readability/informativeness.

The GREEDY algorithm remains the strongest in terms of raw effectiveness (RR) and in either *poisson* (suffixed by _P) or *fixed* set (suffixed by _F), our proposed methods RL rank second after

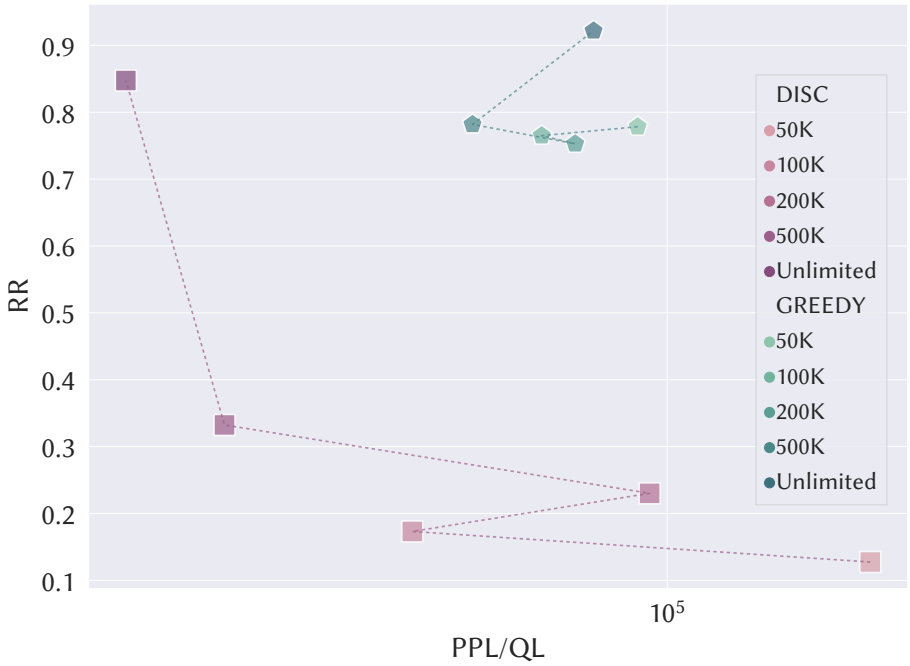


Figure 4.6: Effectiveness for DIS_P and GREEDY of different vocabulary sizes.

the DIS approach for effectiveness. This trend is consistent with experiments on the complete holdout set. Importantly, we can also see from the results of UniLM and T5 that highly readable and informative summaries are not necessarily effective queries. The natural language quality of these two systems was the highest in Table 4.3, but the effectiveness shown here is lower than methods that are optimized directly or indirectly for effectiveness. The comparison of RL against UniLM and T5 further illustrate the importance of the proposed second stage reinforcement learning training for balancing effectiveness and natural language quality.

We plot each system's effectiveness and the aggregation of readability and informativeness score in Figure 4.7. Figure 4.7.1 illustrates the tradeoff between effectiveness and human evaluated readability and informativeness. PSG_F, PSG_P, T5, and UniLM are the clear winners on readability and informativeness but all are clearly less effective. Similarly, the GREEDY and DIS approaches achieve high effectiveness but were determined to be less readable by human assessors. Our system sits in a more balanced position. Figure 4.7.2 illustrates effectiveness versus PPL/QL. Figure 4.7.3 extends human evaluation to Rouge-2, as we found Rouge-2 is correlated with human evaluation with a Pearson correlation coefficient of 0.64. The correlation can also be visually observed in the figure. Figure 4.7.4 confirms that the trends for ROUGE-2 on the human assessed documents are consistent with those computed for the complete holdout set, suggest-

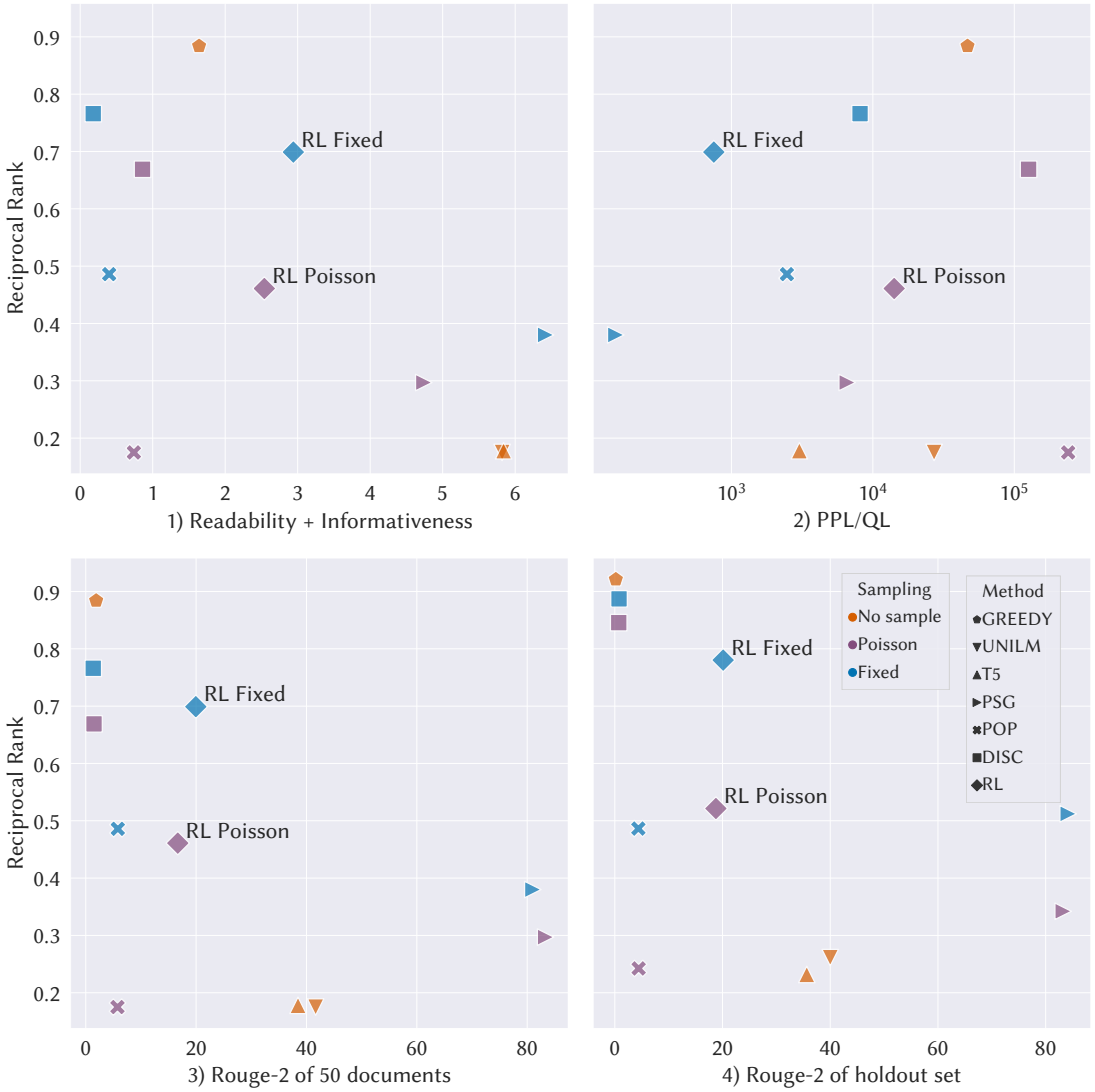


Figure 4.7: Effectiveness-readability/informativeness tradeoff according to human assessments. Subfigure 1 shows results from human assessments. Subfigure 2, 3, 4 shows results from automatic metrics which have a similar trend to human assessments. For subfigure 1, 3, 4, we prefer a system located in the top-right. For subfigure 2, we prefer a system located in the top-left.

Table 4.3: Natural language evaluation for the 50 documents assessed by humans, where † signifies significance at $p \leq 0.05$ in a paired t-test relative to our methods RL_P or RL_F. Our approaches are shown in bold face.

	PPL/QL	ROUGE Precision				Readability	Human Evaluation	
Model	Avg	1	2	L	W	Flesch	Read	Info
GREEDY	46 718.0	100.00	1.89	72.25	59.83	6.65	1.20†	0.44†
UniLM	27 101.8	82.84	41.66	82.23	66.07	66.90	3.06†	2.76†
T5	3023.2	82.20	38.47	80.67	63.96	72.45	3.36†	2.48†
PSG_P	6588.4	93.19	83.34	92.76	81.91	70.51	2.82†	1.92†
POP_P	240 609.6	85.83	5.75	72.13	55.61	71.69	0.62†	0.12†
DIS_P	126 284.6	79.05	1.50	57.05	43.89	24.49	0.62†	0.24†
RL_P	14 152.6	98.14	16.68	81.60	63.92	66.96	**1.74**	**0.80**
PSG_F	152.2	92.13	81.03	90.61	78.98	68.52	3.40†	3.02†
POP_F	2467.5	84.84	5.81	65.28	45.51	69.37	0.34†	0.06†
DIS_F	8118.2	71.15	1.36	47.52	32.78	34.34	0.06†	0.12†
RL_F	750.8	96.91	19.95	73.48	52.81	70.05	**1.74**	**1.20**

ing that ROUGE-2 is indeed a reasonable surrogate for a human assessment when attempting to benchmark quality automatically, at least for preliminary testing of a new model.

4.6.4 Qualitative Analysis

We now discuss the results produced for some documents in order to provide a little more intuition into the queries being generated by each approach, that were subsequently evaluated by the crowdworkers.

The queries generated by the six systems are shown in Table 4.4. Document "D133390" is a description of the 19th amendment which grants women the right to vote. Both T5 and UniLM clearly generate high quality text. POP_F and DIS_F, as their names suggest, select popular and discriminative words respectively. Neither model attempts to induce a natural language ordering of the terms selected, making them much more difficult to comprehend. Similarly, GREEDY, a deterministic version of DIS_F, generates the two rarest terms from the document and provides no meaningful information from the original document. RL_F is not fully correct grammatically since there is a bias towards discriminative terms, and so the term-wise probability is skewed when beam search is applied. Nevertheless, semantically correct phrases such as "gave women right" show that the natural language summarizer is having a positive effect. The method also selected keywords such as "susan", a key figure in the movement, and "tennessee", where the final needed approval was passed increasing the informativeness scores while also being discriminative. In other examples, we can see a similar trend. When compared to T5 and UniLM, our method

Table 4.4: Query examples for documents in the MS-MARCO test collection

System	Query
	Query examples for document D133390
RL_F	susan the tennessee civil constitution movement gave women right
T5	why was the 19th amendment important
UniLM	how did the 19 th amendment change voting rights for women
POP_F	the last became was illegal under state th get
DIS_F	leser schenck the commentscomment symbolic lawmaker electorate cady this
GREEDY	leser commentscomment
	Query examples for document D143592
RL_F	sony full frame produce at 400 plus mm lens but
T5	difference between full frame and aps-c
UniLM	difference between full frame and aps - c
POP_F	full and many are the com it sony lens there
DIS_F	tcav new normal that or dimmest become founder by were
GREEDY	tcav luminance
	Query examples for document D2989514
RL_F	microsoft visual studio code is close to community operation
T5	what is a linq
UniLM	. net framework extensions
POP_F	preview technology the center all data close laptop visual
DIS_F	nowmicrosoft and water fashion tae assume language tree fodmap
GREEDY	nowmicrosoft extension

sometimes sacrifices grammatical correctness and selects “higher impact” terms for the ranking model being used – increasing the likelihood of refinding the document in the collection.

In summary, the best-known techniques such as T5 and UniLM are consistently good at generating human-readable text that accurately represent the source document, while our proposed algorithmic framework retains important properties from summarization but also has much better retrieval effectiveness. Some technical limitations of our proposed method can also be observed in the queries being generated — some readability is being sacrificed in the second stage in order to improve ranking effectiveness, causing the generated queries to be less syntactically complete. In future work we intend to continue exploring how both of these important objectives might be directly optimized simultaneously, further improving our ability to find the more desirable trade-offs between these two competing objectives.

4.7 Summary

In this chapter, we formalize and study the SNLQ problem, which imposes a readability objective in addition to the effectiveness objective on the query generation task. In contrast, the Relevance Modeling technique discussed in Chapter 3 aims at effectiveness only. The readability objective poses new challenges on both ends of the generator: input and output. For the input, the generator needs to understand more complex word dependencies; for the output, the generator needs to generate human-readable queries. By combining state-of-the-art abstractive summarization with reinforcement learning, we can learn a Transformer model for both readability and effectiveness objectives. We show our approach compared against the algorithmic solutions to the strong query problem and the state-of-the-art Transformer-based summarizers. Our approach outperforms the algorithms solutions in readability while still achieving high effectiveness. We also find summarizers produce high-quality summaries but fall short in ranking effectiveness. Our approach is capable of generating queries that balances these competing objectives of ranking effectiveness and human readability.

The approach we introduce in this chapter trains a Transformer model for summarization and ranking in two separate stages. The knowledge learned in the first stage is transferred to the second stage, so the model is able to generate natural language queries while being optimized for ranking effectiveness. In the next chapter, we will discuss a joint modeling approach which combines query generation and ranking. Through Multi-Task Learning, the knowledge of query generation can be transferred to ranking and to further improvements in ranking quality.

5 Enriching Ranking Models using Query Generation

5.1 Introduction

Up to this point, we have discussed using Relevance Modeling to improve a query, and using Transformers to generate effective queries. They both try to improve the information need representations from a query perspective. In this chapter, we present a method of improving the understanding of queries from the ranking model perspective. Specifically, we incorporate query generation into the training process of a ranking model, so the model is enriched with extra relevance signals and is more effective and generalized.

Transformers not only are effective at generation but also have shown promising potential in ranking, in which the language modeling pretraining plays a critical role. Transformer based models are often pretrained with language modeling tasks to obtain general language knowledge, then they can be easily finetuned for specific tasks, such as generation and ranking. The language modeling task focuses on general language understanding without a focus on relevance — the relationship between a document and a query. This decouples specific tasks from the general pretrained models, so these models can be very versatile. This pretraining approach also opens a gap. Can we further enrich its language capability with a stress on relevance? We will present a method that can exploit known query-document pairs and incorporate such relevance signals into ranking models, i.e. we jointly train ranking and query generation.

While a Transformer architecture can be used for ranking and query generation independently, little prior work has focused on combining the two into a single model, and the connection between them can be also seen from a probabilistic perspective. Developing efficient and effective retrieval models has been at the core of information retrieval (IR) research since the 1960s [46]. Early models, such TF-IDF [180], relied on heuristics derived from statistical properties of documents contained in a collection. The well-known *probability ranking principle* (PRP) of Robertson [168] provided a theoretical foundation to these early ideas using probability theory. PRP frames a retrieval model as being an estimate of the probability of a document D being relevant (R) to the query Q, i.e., $P(R|Q, D)$, where R is a binary random variable. This probability

can be estimated using a *discriminative model* or a *generative model*. The terms *discriminative model* and *generative model* map to a transformer ranking model and a transformer generation model respectively. We will use *transformer ranking/generation model* when we discuss the specific models used in this chapter and use *discriminative* and *generative* when we discuss the more generalized concepts.

Discriminative models for retrieval were first proposed by Fox [67] and later adopted in a number of successful learning to rank models, including SVM-based and decision tree-based modeling approaches [27, 89]. More recent neural ranking models in this category have also been proposed [56, 76, 145]. Generative models have also been used by a number of well-known retrieval models. For example, classic probabilistic retrieval models such as BM25 [166] are based on a document generation assumption. In contrast, the statistical language modeling, which is the basis of models such as query likelihood [160], are based on a query generation assumption.

Therefore, it is clear both discriminative and generative approaches have merits, and can potentially surface valuable signals which can be used to improve retrieval effectiveness. In this chapter, we show how both of these approaches can be used together to produce more effective retrieval models. More specifically, we are interested in resolving the following hypothesis:

Joint discriminative and generative retrieval modeling leads to more generalized, and hence more effective retrieval models.

To validate our hypothesis, we introduce a novel Multi-Task Learning (MTL) framework in which the tasks include both discriminative and generative relevance models. In the discriminative tasks, the model maximizes the likelihood of predicting the relevance labels given queries and documents, while in the generative tasks, the model is optimized to generate queries/questions given documents.

Developing a neural network architecture for both discriminative and generative modeling of relevance is challenging, because generative models rely heavily on encoder-decoder based architectures, while discriminative models typically require only an encoding component. We investigate two approaches to joint modeling: (1) an encoder-only architecture in which the generative tasks are modeled using input masking. For example, query generation is modeled by masking the query in the input of the network and using a maximum likelihood objective to predict the query tokens associated with the masked tokens; (2) an encoder-decoder Transformer architecture [201] in which the discriminative relevance modeling is implemented by feeding documents and queries to encoder and decoder inputs respectively, and using the cross-attention mechanism between encoder and decoder components (also known as the encoder-decoder attention) to learn a relevance score for a query-document pair.

In order to resolve our challenges, we have focused our work on the following research questions:

RQ1 Do generative tasks improve discriminative retrieval models?

RQ2 Which neural network architectures (i.e., encoder-only or encoder-decoder) produce more effective joint discriminative and generative relevance models?

RQ3 Are the resulting models easily transferable to other retrieval tasks?

In summary, our experimental results support our hypothesis that discriminative neural ranking models can be generalized using generative tasks and importantly, does not rely on a specific architectural configuration. This generalization leads to significant improvement across models and collections for both precision- and recall-oriented metrics. Our experiments also highlight the significant impact of such a joint modeling on other task-transfer scenario.

5.2 Background

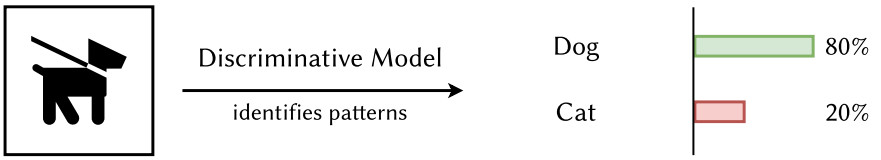
5.2.1 Generative and Discriminative Models

The key motivation for using query generation to improve ranking models is the strong connection to the concept of *discriminative* and *generative* models. The terminology is better explained through examples. Consider the task shown in Figure 5.1 which aims to classify dogs and cats. In Figure 5.1a, a discriminative approach, similar to a human, fulfills the task by showing its answer directly — the probability of the image being a dog is 0.8 and thus the answer is "dog". A discriminative model often achieves the answer by identifying discriminative patterns in the image. A generative approach is more indirect. It first tries to draw pictures of a dog and a cat (thus "generative"), and asks "What are the chances of drawing a dog/cat like the target image?", as shown in Figure 5.1b. Since drawing a dog is more likely to result in the target picture, the mode is more confident that "dog" is the correct answer. This is not very intuitive from a human perspective. Why do we make the distinctions? The reason lies in how we factor the task using probability theory, which we will further discuss in the next section.

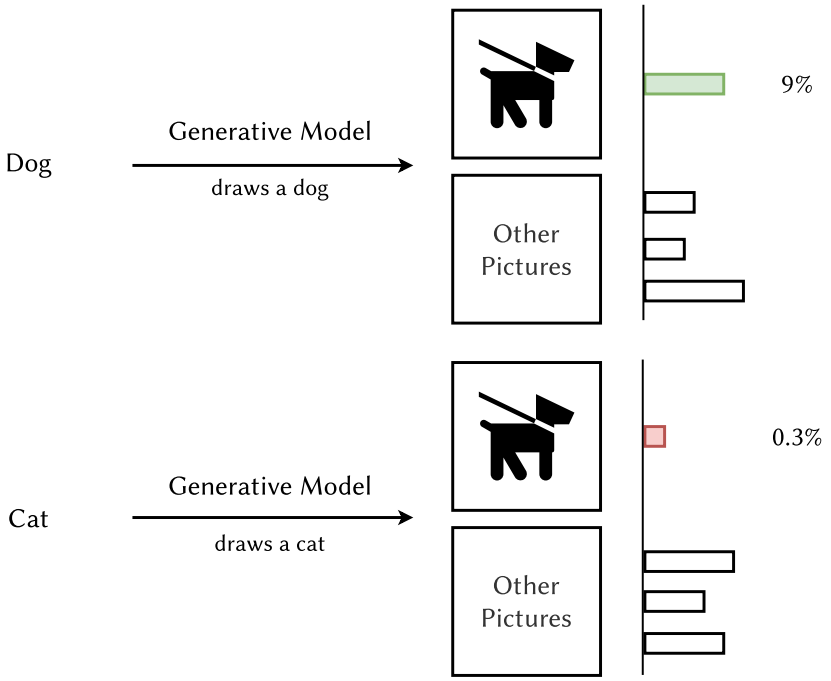
However, while drawing a dog or a cat is hard, the process of learning to draw makes us understand better about the characteristics of a dog and a cat, and reinforces our knowledge in the long run, or in other words, results in more generalized models. We believe this intuition applies to discriminative and generative models as well. Discriminative models are powerful when numerous training instances are available, but the signals contained in the generation process is also important.

Key Concepts. Discriminative and generative approaches are two of the fundamental methods used in statistical classification problems. As the name suggests, discriminative models discriminate data instances directly. They usually produce probabilities of data instances belonging to pre-defined classes. Generative models assume data instances are generated from given labels and convert the generation probabilities into classification probabilities through Bayes' Rule.

More formally, given a data instance X and a label Y, the classification objective is to predict the probabilities of the data instance belonging to a label $P(Y|X)$. Discriminative models can estimate this probability directly without further factorization. Generative models factor $P(Y|X) \propto P(X|Y)P(Y)$, a conditional probability $P(X|Y)$ and a prior probability $P(Y)$. Generative models differ in the assumptions they make and how they estimate the factored probabilities.



(a) A discriminative approach.



(b) A generative approach.

Figure 5.1: Discriminative and generative models in a classification task. The probabilities shown are artificial for illustration purpose.

This distinction can have a significant empirical impact in an application. It is more challenging for generative models to precisely model the underlying distribution as it is maybe harder to learn to draw a dog than telling the difference between a dog and a cat. Generative models also make more assumptions in the intermediate steps to calculate probabilities, which can limit accuracy especially when training data grows [46]. In practice, discriminative models are often more favorable due to these reasons. In the machine learning domain, it has also been shown that the discriminative models have lower asymptotic error as training data grows [142].

Generative and Discriminative Retrieval Models. We have reviewed retrieval models in Sec 2.2. We can also categorize the probabilistic models into generative and discriminative classes.

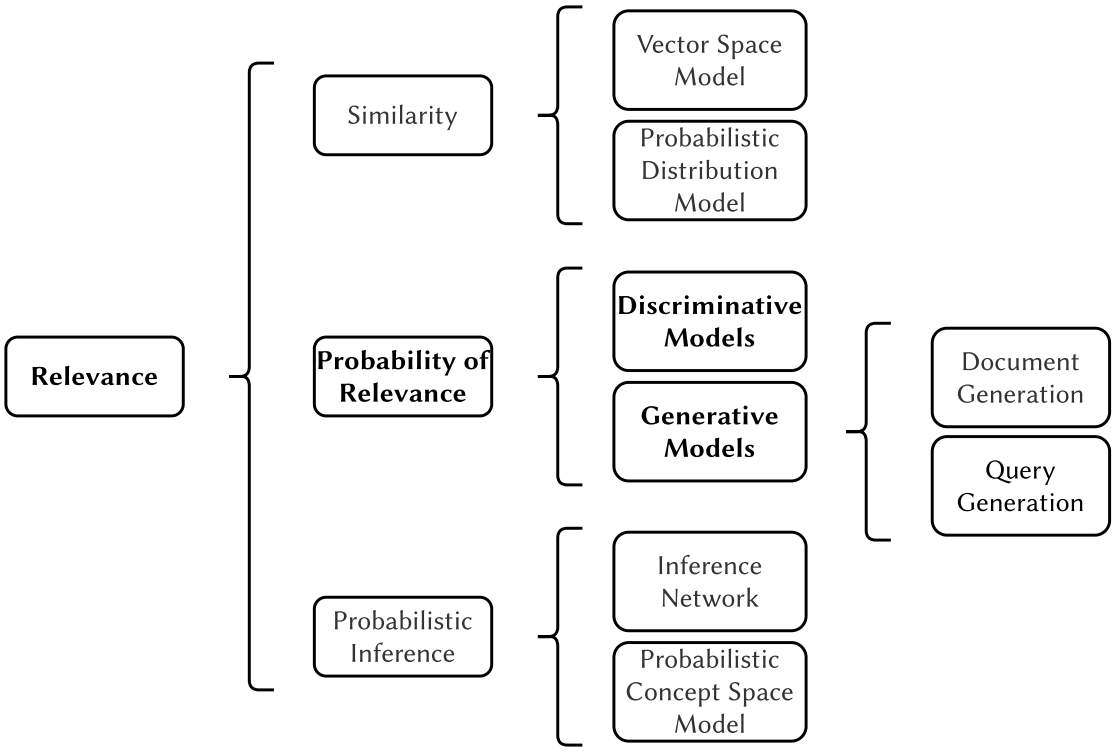


Figure 5.2: Discriminative and generative models in the context of relevance estimation. (Image source: adapted with modification from https://people.cs.umass.edu/~zamani/cs646.)

This perspective also allows us to understand their advantages and disadvantages as have been studied in machine learning.

Figure 5.2 shows the generative and discriminative methods in the context of relevance estimation. Early classic retrieval models are mostly generative models [139], as in the early days the resource constraints (e.g. computing resource and training data) limits our ability to train a high-accuracy discriminative retrieval model. Examples are the Binary Independence Retrieval model [170], a Two-Poisson model [172], and Language models [160]. The Language Modeling approach is arguably one of the most popular due to its theoretical foundations and empirical success. Nowadays, Language Modeling is still one of the key components in the state-of-the-art Transformer-based retrieval models.

Discriminative retrieval models, as described before, take a direct approach in estimating relevance. These models often take as input a query feature vector and a document feature vector and directly produce the relevance probability. Typical examples are the maximum entropy model [21], Rank-SVM [89], LambdaMART [27], and more recently neural ranking models [145]. Neural ranking models have been reviewed in Sec 2.3.

The properties of generative and discriminative retrieval models are summarized in Table 5.1. Discriminative models are motivated by learning data patterns to distinguish them while gen-

Table 5.1: Comparison of discriminative and generative retrieval models.

	Discriminative	Generative
Motivation	Distinguish data	Generate data
Factorization	$P(R\|D,Q)$	$P(R\|D,Q) \propto P(D,Q\|R)P(R)$
Relevance Estimation	Direct	Indirect
Modeling Assumptions	None	Have intermediate assumptions
Training Data Requirements	High	Low[46]
Expressiveness[139]	High	Low

erative models assume data are generated and use a more indirect approach. This difference is articulated by Vapnik [200]: "…try to solve the problem directly and never solve a more general problem as an intermediate step …" Discriminative models can also easily incorporate arbitrary features and automatically learn features such as word embeddings, which makes them more expressive and flexible. However, discriminative models are highly dependent on training data. Prior to the recent growth in neural ranking techniques, generative models have enjoyed greater empirical success, and were valuable tools that capture relevance signals.

Generative Adversarial Networks. Most retrieval approaches focus primarily on improving ranking effectiveness by optimizing either a discriminative or a generative retrieval model. IRGAN [207] also includes a discriminative and generative component which improves retrieval effectiveness using a Generative Adversary Network (GAN) [74]. The IRGAN model formalizes the retrieval problem as a min-max game where the discriminative and generative models compete with each other – the generative model estimates the relevance distribution over signals using the discriminative model and then the discriminative model uses the output from the generative model to produce a better estimate of document rankings. In the follow-up work, Zou et al. [228] derive a theoretical analysis that connects the query reformulation and the document ranking problem using a game-theoretical approach, which models the two tasks as a general-sum game and a partnership game, respectively. Although related, this line of work differs from ours in two key aspects: first, we rely primarily on multi-task learning where auxiliary tasks are often considered as complementary to the main task, instead of a "competitor" in a GAN; second, tasks in GAN-based approaches are strongly coupled. The discriminator must be carefully designed to compete against the generator and the tasks involved are restricted to two due to the design (one for discriminator and one for generator). In our framework, any number of loosely related tasks can be easily incorporated to improve the main task.

5.2.2 Multi-Task Learning

Multi-Task Learning, as its name suggests, jointly learns multiple tasks. It is motivated by the intuition that related tasks have mutually beneficial training signals. Formally, given a set of

tasks $\mathcal{T} = \{T_1, T_2, \dots\}$ and a set of loss functions $\mathcal{L} = \{L_1, L_2, \dots\}$, the objective of MTL is to minimize the joint loss

$$\min_{\theta^{sh}, \theta^1, \dots, \theta^T} \sum_{i=1} \lambda_i L_i(\{\theta^{sh}, \theta^i\}, T_i)$$

where λ_i is the weight of each loss L_i, θ^{sh} is the shared parameters and θ^i is the independent parameters of each task. This formulation highlights two major design choices in MTL: network architecture (θ^{sh} and θ^i) and task balancing (λ_i) [118]. Network architecture is mainly about sharing: "what to share" and "how to share" [225]. Task balancing focuses on the weights of task losses. Learning related tasks together may help transfer the knowledge between tasks and improve the overall performance. Recent pretrained language models have benefited from multi-task learning.

The network architecture decides which part of the network is shared among tasks and which are independent of each other. In fact, popular Transformer models BERT [57], BART [111], and T5 [164] are all MTL models which introduced a common multi-tasking practice of adding new heads on top of the underlying pretrained language model. When we apply MTL in this work, we also follow this approach which is illustrated in Figure 5.4 and Figure 5.6 when we discuss our methods.

Task balancing decides how each task contributes to the optimization of the model. The simplest approach is using equal weights which is adopted by BERT [57], BART [111], and T5 [164]. Other approaches, according to Navon et al. [141], include GradNorm [32], Gradient Cosine Similarity (GCS) [62], Uncertainty [97], and Dynamic Weight Averaging (DWA) [118]. These methods are reported to be superior to others in different scenarios, but it is unclear which is the best one for the ranking task. The Uncertainty method showed promising results in our preliminary experiments and is simple and efficient compared to other adaptive approaches and thus is used in this work. Exploring the weighting methods for IR tasks is another interesting line of research but orthogonal to this work.

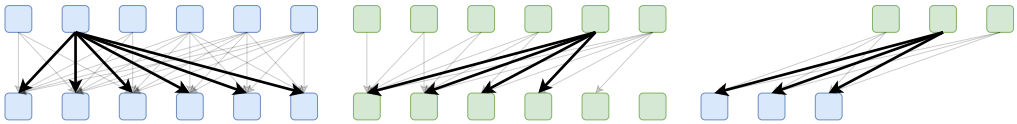
Multi-Task Learning in IR. Multi-task learning is another cornerstone of this chapter. It is concerned with learning multiple related tasks jointly while transferring the common knowledge across tasks. A large body of work explores applying multi-task learning (MTL) to retrieval tasks. Broadly speaking, these approaches can be categorized into two types. Models in the first category tries to learn similar functions for multiple tasks. For example, Nishida et al. [144] consider both extractive question answering and document ranking tasks, which can both be cast to classification tasks; both recommendation and retrieval tasks formulated by Liu et al. [120] are ranking tasks; Salehi et al. [178] combined semantic classification and query segmentation. Studies in the second category learn common representations and adapt the learned model to other domain-specific problems or heterogeneous tasks. For example, Bai et al. [7] leverage the MTL approach to learn "super features" which can be applied to search tasks in different domains; the method proposed by Liu et al. [121] learns a general representation by jointly learning from both query classification and document ranking tasks; work has also been done in session-based

retrieval [2, 3], where the goal is to represent query, document and users' context using MTL. In this work, the core idea is to select supplementary tasks from one category of tasks (generative) and use it to improve models in another category (discriminative). More specifically, our main focus is the *retrieval model* instead of representation learning, which may enable much more flexibility when selecting auxiliary tasks to complement the model being targeted.

5.2.3 Attention: A Different Perspective

We introduced the generalized attention mechanism in Sec 2.4.1 but did not discuss the difference in *self-attention* and *cross-attention* which has important implications on dependency modeling. Now, we briefly revisit the attention mechanism from the dependency modeling perspective.

Encoder: Bidirectional Attention. An encoder transformer is broadly used for classification, regression, and other related tasks. The core component of an encoder is the bidirectional attention mechanism as illustrated in Figure 5.3a. Using ranking as an example, when we feed the concatenation of a query and a document into an encoder, token embeddings are updated according to the context of the entire sequence. Bidirectional attention can take full advantage of the information not only within a query and a document but between them.



(a) Bidirectional **(b)** Unidirectional **(c)** Cross

Figure 5.3: An encoder implements bidirectional attention. A decoder implements unidirectional attention and cross attention. Cross attention bridges an encoder to a decoder.

Decoder: Unidirectional and Cross Attention. Encoder-decoder transformers have decoders in addition to encoders. The encoder-decoder transformers are primarily used in seq2seq tasks which is an abstraction that maps one sequence of text to another. Examples are summarization, machine translation, query generation, and question answering. A source sequence is fed into the encoder. Then the decoder regressively produces new tokens conditioned on the source and on past tokens that were generated. The generation process often repeats several times until an end-of-sentence identifier is encountered or a pre-defined length limitation is reached. Formally, given the source sequence X and a target sequence Y, the probability distribution of the next token is conditioned on X and past tokens $Y_{<i}$.

$$P(Y|X) = \prod_{i}^{|Y|} P(y_i|Y_{<i}, X) \tag{5.1}$$

The regressive nature of the model is reflected in the attention design of the decoders. Within a decoder, the self-attention is causal, or unidirectional, as in Figure 5.3b. A token can only attend to past tokens. This property corresponds to conditioning on $Y_{<i}$ in Eq 5.1. Following the unidirectional attention is cross attention (Figure 5.3c) where the model conditions on the source input X as in Eq 5.1. The various attention mechanisms illustrate the fundamental difference between an encoder transformer and a decoder transformer.

5.3 Joint Discriminative and Generative Retrieval using MTL

In this section, we introduce our GDMTL (Joint Discriminative and Generative Retrieval Model with Multi-Task Learning) framework for improving discriminative retrieval models using generative tasks in Sec 5.3.1, then in Sec 5.3.2 and Sec 5.3.3 we describe how we implement each component of that general framework using Transformer networks and unify the different requirements of discriminative and generative models.

5.3.1 GDMTL Framework

The GDMTL framework combines the ideas of discriminative and generative retrieval models using multi-task learning. In more detail, the framework simultaneously optimizes two different objectives: (1) one objective for a discriminative retrieval model, and (2) one set of objectives for generative tasks. GDMTL may contain multiple generative tasks. In the next section, our experiments will provide empirical evidence that two generative tasks provides better generalization for the discriminative retrieval model. We now formalize the GDMTL framework.

Problem Formulation. Let $\mathcal{T} = \{(Q_1, \mathcal{D}_1, \mathcal{R}_1), (Q_2, \mathcal{D}_2, \mathcal{R}_2), \cdots, (Q_n, \mathcal{D}_n, \mathcal{R}_n)\}$ be the training set for a ranking task with n training queries, in which Q_i denotes the i^{th} query, and $\mathcal{D}_i = \{D_{i1}, D_{i2}, \cdots, D_{im_i}\}$ denotes the set of documents for Q_i in the training set. $\mathcal{R}_i$ is a set containing ground truth relevance judgments, such that $\mathcal{R}_{ij}$ represents the relevance judgment for the query-document pair of (Q_i, D_{ij}). This task trains a ranking model that re-ranks the documents in a given candidate document set $\mathcal{D}'$ for a test query Q'.

GDMTL Overview. The GDMTL framework consists of a discriminative retrieval model M_d (parameterized by θ_d) and a number of generative models $M_g : g \in G$ (each parameterized by θ_g), where G is a set of all generative tasks that enrich the discriminative retrieval model. Without loss of generality, we assume that relevance labels are binary, thus M_d estimates the probability of the document $D_{ij} \in \mathcal{D}_i$ being relevant to the query Q_i using $P(R = 1|Q_i, D_{ij}) = M_d(Q_i, D_{ij}; \theta_d)$, where R is a binary random variable. This can be easily extended to graded relevance labels. With the ground truth relevance labels R_i, the parameters θ_d are learned by minimizing a loss $L_d(\theta_d; Q_i, \mathcal{D}_i, \mathcal{R}_i)$. The total loss function is computed by averaging L_d for all queries in $\mathcal{T}$.

In contrast, each generative model $M_g : g \in G$ estimates the probability of a target text T being generated from a source text s: $P(T|S; \theta_g) = M_g(S; \theta_g)$. The parameters θ_g are opti-

mized using a different loss $L_g(\theta_g; T, S)$. The question now is: How are T and S related to the training set $\mathcal{T}$? One reasonable approach, which is mainly used throughout this chapter, is to use relevant documents as source texts and queries as target texts. This casts the problem to a query generation retrieval model, similar to query likelihood [160] and doc2query [146]. In summary, one way of modeling this component is to compute the query generation probability $P(Q_i|D_{ij}, R_{ij} = 1; \theta_g) = M_g(D_{ij}; \theta_g, R_{ij} = 1)$. Thus, the loss function of this part would be $L_g(\theta_g; Q_i, D_{ij}, R_{ij})$. As discussed previously, we have designed GDMTL such that it can optimize multiple generative tasks. Multiple generative tasks in addition to query generation can be used, such as question generation based on QA data (e.g., generating questions from answer documents) and anchor text generation based on a hyperlink graph from Web data.

Using multi-task learning, parameters are split into shared and independent model-specific parameters. Therefore, θ_d and $\theta_g : g \in G$ share the parameters θ_{sh}. The resulting loss function is:

$$L = w_d L_d(\theta_d) + \sum_{g \in G} w_g L_g(\theta_g) \tag{5.2}$$

where w_d and $w_g : g \in G$ are the weights assigned to the respective losses.

Implementing the Loss. The discriminator loss L_d can be modeled as either a point-wise, pair-wise, or list-wise loss function. Without loss of generality, in our experiments, Hinge loss is used for pair-wise ranking. For a pair of document candidates $D_{ij}, D_{ik} \in \mathcal{D}_i$ for the query Q_i, the discriminator loss function is defined as follows:

$$\begin{aligned} &L_d(\theta_d; Q_i, \mathcal{D}_i, \mathcal{R}_i) \\ &= \frac{1}{Z} \sum_{\substack{1 \le j,k \le m_i \\ r_{ij} \neq r_{ik}}} \max \big\{0, \epsilon - \text{sign}(R_{ij} - R_{ik}) \\ &\qquad\qquad (M_d(Q_i, D_{ij}; \theta_d) - M_d(Q_i, D_{ik}; \theta_d))\big\} \end{aligned} \tag{5.3}$$

where Z is a normalization factor and is equal to the number of training pairs for the query q_i. ϵ is a hyper-parameter for the Hinge loss and is set to 1 for binary relevance labels.

The generative loss $L_g : g \in G$ is defined using a cross entropy loss function as in the seq2seq model [194], and is equivalent to maximizing the likelihood of generating the target sequences observed in the training data. The generative loss for a query generation task is:

$$L_g(\theta_g; Q_i, \mathcal{D}_i, \mathcal{R}_i) = -\frac{1}{Z'} \sum_{\substack{1 \le j \le m_i \\ R_{ij}=1}} P(Q_i|D_{ij}, R_{ij} = 1; \theta_g) \tag{5.4}$$

where Z' is a normalization factor and is equal to the number of relevant documents for the query Q_i. In the above loss function, $P(Q_i|D_{ij}, R_{ij} = 1; \theta_g)$ is computed as follows:

$$P(Q_i|D_{ij}, R_{ij} = 1; \theta_g) = -\sum_{t=1}^{|Q_i|} \log P(Q_i^t|Q_i^1, Q_i^2, \cdots, Q_i^{t-1}, D_{ij}) \tag{5.5}$$

where Q_i^t denotes the t^{th} token in the query. This loss function estimates the likelihood of each target token being generated given the input and all the previous tokens in the ground truth. These probabilities are produces using the underlying model M_g.

Balancing Loss. As shown in Eq (5.2), the contributions of each task on updating the shared parameters θ_{sh} is controlled by the weights w_d and $w_g : g \in G$. A straightforward approach to assign loss weights is to treat them as hyper-parameters and tune them on a held-out validation set. However, it is expensive and sometimes impractical to exhaustively explore the parameter space when the training cost is high and there is more than one task, which reinforces the importance of automatically learning these parameters.

In this work, the Uncertainty [97] weighting scheme is used to automatically learn the weights. This method models the *homoscedastic* uncertainty of tasks, which represents the confidence in the contributions from different tasks and is independent of the input data. Intuitively, if a task has high uncertainty, our confidence is lower, and therefore the contribution to the joint loss is reduced, and so on. We adopt a variation of the Uncertainty method by Liebel and Körner [113] which has a modification to the regularization to avoid negative loss values. This makes Eq 5.6:

$$\begin{aligned} L_{\text{uncertainty}} =& \frac{1}{2\sigma_d^2} L_d(\cdot) + \log(1 + \sigma_d^2) + \\ & \sum_{g \in G} \frac{1}{2\sigma_g^2} L_g(\cdot) + \log(1 + \sigma_g^2) \end{aligned} \tag{5.6}$$

where σs are learnable parameter modeling the uncertainty of the models. When the loss is high, the uncertainty is high (but penalized by $\log(1+\sigma_\cdot^2)$ for becoming too high), leading to low contribution of the loss, avoiding the gradients of this model dominating the training, and vice versa. Please refer to Kendall et al. [97] for further rationale in these formulations.

5.3.2 Architecture I: Encoder-Only GDMTL

The proposed GDMTL framework can be implemented using two different general architectures. The architectural choice of multi-task learning models affects how the shared parameters are optimized. In this subsection, we show how to implement M_d and M_g in a unified encoder architecture. M_d can be implemented using a neural network encoder that learns a representation for a query Q_i and a document D_{ij}. This representation is then used to produce a relevance score for the given query-document pair. For example, a common approach is to concatenate query

and document tokens using a beginning and separation token and feed it to a pretrained BERT variant [57] (i.e., the encoder). The representation produced by BERT for the beginning token is then fed to a fully-connected layer to output the relevance score [145, 153].

Challenges in Modeling Seq2seq Using Encoders. Although an encoder-only architecture fits well with a discriminative ranking model, modeling generative tasks with no decoder is not straightforward since there is no longer an autoregressive component for text generation. To address this issue, we adapt the idea of predicting masked input, similar to that of masked language model (MLM) training as in BERT [57]. More specifically, M_g can be modeled by masking all input query tokens and using the associated output representations to predict them. However, there is an important limitation when using MLM – it does not scale when using long spans. As a point of reference, Joshi et al. [92] proposed spanBERT which extends BERT by masking contiguous random spans. In their work, the mean span length is 3.8 tokens. However, the average number of words (before WordPiece tokenization) of MS-MARCO training queries is 7.4. We have conducted experiments using MLM and found that training converges prematurely, and has reduced overall performance. The limitations of this approach for our task is intuitive in retrospect. Casting query generation to the MLM task is akin to asking the model to generate a few tokens without having access to the previous tokens that were generated. This can be further improved by combining ideas from seq2seq and MLM training. However, we cannot fully adapt a seq2seq training strategy because of the bidirectional attentions in BERT layers; otherwise, computing the loss function for generating each token would depend on the future tokens to be generated. A solution to this problem is to convert each text generation training instance to t training instance, where t denotes the number of tokens in the target text. In this case, the first training instance has all the masked tokens for all target inputs and the loss function is only computed when generating the first token; the second training instance, on the other hand, would have the first target token as input with masked tokens for the rest and the loss function would be only computed for generating the second token; and so on.

We now present a new attention mechanism we call *mixed attention*, that theoretically produces in the same loss and gradient values, but is easier to train in practice.

Mixed Attention. To overcome the aforementioned difficulties, we propose a mix of bidirectional attention, unidirectional attention, and cross attention to support seq2seq tasks in our encoder architecture. Figure 5.5a shows how mixed attention combines the three attention mechanisms into one: the document tokens (light blue) can fully attend to themselves (bidirectional attention), the query tokens (light green) can attend to the past query tokens (unidirectional attention), and the document tokens (cross attention).[1]

[1]The proposed mixed attention can be considered as a special case of the masked attention as used in other work such as Zeng and Nie [222]. We would like to note that the proposed mixed attention was developed independently of other work.

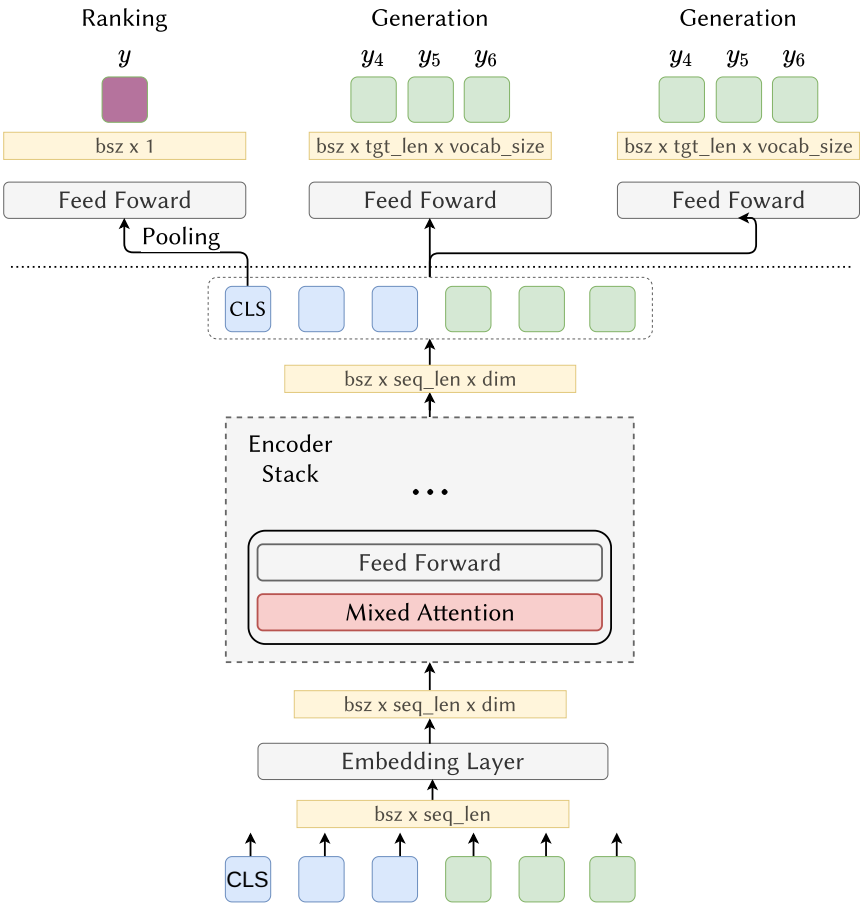


Figure 5.4: The MTL encoder incorporates generation tasks using mixed attention. The model fully attends a document but only attends backwards for a query, imitating the behavior of a decoder. *bsz*: batch size; *tgt_len*: target sequence length; *seq_len*: total sequence length; *dim*: embedding size; *vocab_size*: vocabulary size.

Formally, let D denote a document, the contextualized embeddings of token t of a document-query pair after mixed attention can be expressed as follows:

$$z_t = \begin{cases} \sum_{i=1}^{|D|} \frac{\exp(q_t^* k_i^*)}{\sum_{j=1}^{|D|} \exp(q_t^* k_j^*)} v_i^*, & \text{if } t \leq |D| \\ \sum_{i=|D|+1}^{t-1} \frac{\exp(q_t^* k_i^*)}{\sum_{j=1}^{t-1} \exp(q_t^* k_j^*)} v_i^*, & \text{otherwise} \end{cases} \tag{5.7}$$

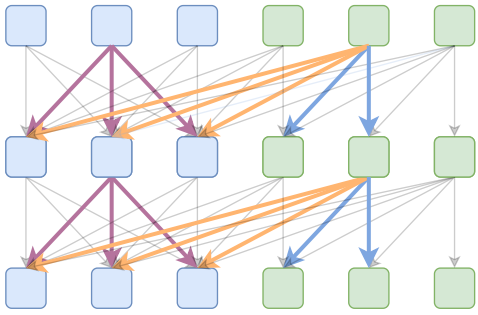
where q^*, k^*, and v^* represent query, key and value vectors of the attention mechanism, respectively.

Mixed attention can be implemented by modifying the attention masks without changing the vectorized attention computations. This allows us to apply mixed attention to existing pretrained

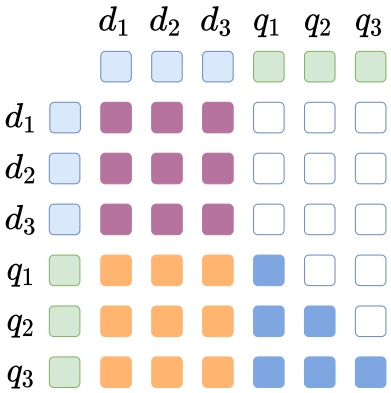
Transformer models. Properties of the special mask is shown in Figure 5.5b. The bidirectional attention mask is a 3×3 all-one matrix (purple), the unidirectional attention mask is a 3×3 lower triangular matrix (blue) and the cross attention mask is also a 3×3 all-one matrix (yellow). Combining them results in the mixed attention mask.

Now we consider how mixed attention can be stacked as transformers often contain multiple attention blocks. One important property of unidirectional attention is that it can be stacked arbitrarily in a decoder transformer without leaking labels. This property is retained in mixed attention, as shown in Figure 5.5a. The query tokens in the top layer represented with light green attend past query tokens in the middle layer which regressively attend solely to past query tokens in the bottom layer. Thus, mixed attention can be used safely in an encoder to imitate complete encoder-decoder behaviors.

Model Implementation using Mixed Attention. Using mixed attention, we can extend an encoder Transformer to implement discriminative and generative retrieval models using multi-task learning, as shown in Figure 5.4. For ranking, the model takes the concatenation of the document and query as input. An embedding produced from the "[CLS]" token is fed into a feed-forward layer to produce a score for the input pair (Ranking Head in Fig 5.4). For query generation, we also concatenate the document and the query as input into the model, but apply mixed attention instead of bidirectional attention to imitate encoder-decoder behavior. This model produces the next token probabilities for a target input using the *Generation* head. The model can easily be extended by adding additional heads, as is now common practice in multi-task learning regimes. We refer to this implementation as GDMTL in later sections.



(a) Mixed Attention of bidirectional (purple), cross (yellow) and unidirectional (blue) attention.



(b) Mixed Attention Mask. Each row represents a token attending to other tokens with filled boxes. For example, d_1 can attend to d_1 to d_3; q_2 can attend to d_1 to q_2.

Figure 5.5: Mixed attention imitates bidirectional, unidirectional and cross attention behaviors.

5.3.3 Architecture II: Encoder-Decoder GDMTL

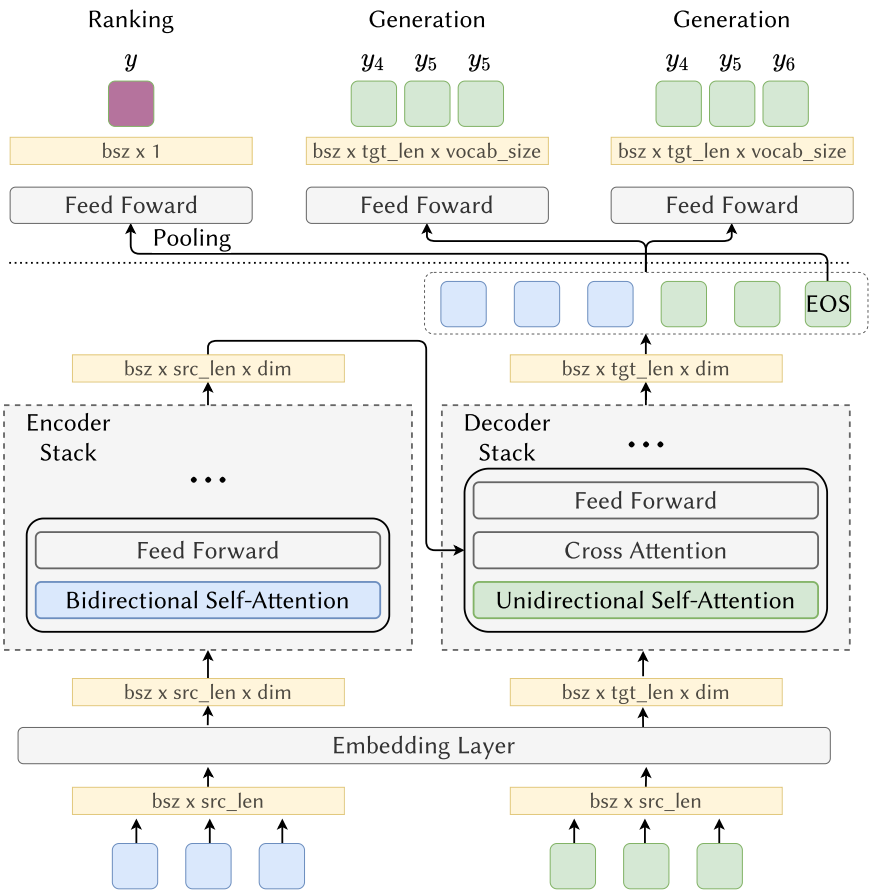


Figure 5.6: An MTL encoder-decoder architecture that combines ranking tasks and generation tasks. *bsz*: batch size; *tgt_len*: target sequence length; *src_len*: source sequence length; *dim*: embedding size; *vocab_size*: vocabulary size.

In this subsection, we present an alternative solution to implement the GDMTL framework using an encoder-decoder architecture. It is less common to use such an architecture for ranking tasks, as encoder-only architectures are reasonable decoders and are most valuable for autoregressive generation tasks where each prediction depends on the previous one. However, unlike an encoder-only architectures, encoder-decoders make it simpler to model generative models M_g.

That said, an encoder-decoder Transformer can also be used for ranking. As discussed in Section 5.2, Raffel et al. [164] feed the same concatenation of the query and the document to both the encoder and the decoder and fine-tune the model to predict "True" and "False" literals.

We also adapt an encoder-decoder Transformer with multiple attention heads, as shown in Figure 5.6. In our proposal, the model produces a score with the *Ranking* head and sequences of generated text with the *Generation* heads. For ranking, the encoder takes the document as input and the decoder takes the query as input, and the prediction of the next token for the entire sequence is fed into a feed-forward layer. Note that, in contrast to an encoder Transformer, the first token cannot be regarded as a complete sentence representation given the decoder limitation of having only unidirectional attention. So, only the last input token has access to the entire sequence, and can be used to fine-tune the model. For query generation, the model produces a distribution over the entire vocabulary for each token in a target instance. Thankfully, we can use pretrained encoder-decoders, such as BART [111], and take advantage of self-supervision pretraining. We refer to this model as GDMTL_S in later sections.

We also propose a variation of BART for ranking. Instead of feeding separate documents and queries into the encoder and decoder, they are concatenated and feed to both the encoder and decoder. Then the model is fine-tuned as have described. This variation is referred to as GDMTL_C in later sections.

5.4 Experimental Setup

5.4.1 Dataset

We use two datasets to evaluate our methods: MS-MARCO and CAsT 2019.

MS-MARCO Passage. For passage retrieval, we use MS-MARCO, which consists of 1 million queries sampled from Bing search logs and 8.8 million passages extracted from web documents [143]. The queries are split into train, dev, and eval sets by the organizers. The training set contains around 532k relevant query-document pairs, the majority of which have only one relevance assessment per query (95% of training queries have a single passage assessment). The dev set contains 6,980 queries, 6,950 of which have one relevant passage. Model evaluation is performed using the dev set and 5-fold cross validation. It is not possible to use the eval set since the judgments are not publicly available.

CAsT 2019. We also use CAsT 2019, which is a conversational search task consisting of two document collections: MS-MARCO and TREC CAR (Washington Post Collection was originally included but not used in the final evaluation by the organizers). Here we use the MS-MARCO subset to evaluate the effectiveness improvement and use the TREC CAR subset to evaluate our model generalizability. The CAsT 2019 test collection contains 20 multi-turn sessions. Within a session, multiple queries are issued for an information need. We use the resolved queries provided by organizers in our experiments since our focus is on the ad-hoc retrieval task, and not co-reference resolution as in the conversational case. The resulting experimental dataset contains 173 queries and 2,983 relevant passages. We perform cross validation at the session level instead of query level as the queries are not i.i.d. due to the intentional session-level dependency. We

use the *GroupKFold* algorithm from *sklearn* [2] to guarantee that queries in the same session are in the same fold. In each evaluation iteration, we use three folds for training, one for validation and one for testing.

MS-MARCO QA. We also use the MS-MARCO QA dataset in the question-answering task as an auxiliary generative dataset. This dataset has overlapping queries and relevant passages as found in the passage ranking task. In addition, it contains human crafted answers derived manually from relevant passages. We use them as the target of the generative task.

5.4.2 Task Setup

Retrieval Task. We use the MS-MARCO and CAsT 2019 dataset for the retrieval task. We use Anserini toolkit for indexing and first-stage retrieval, BM25 tuned for recall based on 1,000 randomly sampled queries from the training set. The MS-MARCO corpus is enriched with DeepCT [50] for first-stage retrieval only. For each query we retrieve 1,000 documents for second-stage re-ranking. All the ranking models we use in this work are summarized in Table 5.2.

Table 5.2: Model notations.

Notation	Description
mono	monoBERT ranking model from Nogueira and Cho [145] and Nogueira et al. [147]
BERT	BERT ranking model (our implementation)
GDMTL	Our multi-task encoder model: ranking and query generation
GDMTL+	Our multi-task encoder model with 3 tasks: ranking, query generation, question answering
BART_S	BART ranking model using *separated* passage and query input
GDMTL_S	Our multi-task encoder-decoder model using *separated* passage and query input
BART_C	BART ranking model using *concatenated* passage and query input
GDMTL_C	Our multi-task encoder-decoder model using *concatenated* passage and query input

Generative Tasks. We now explore the use of two auxiliary tasks, which are query generation and question-answering. For the auxiliary query generation task, we take advantage of known

[2] https://scikit-learn.org

query-document pairs using qrel data. That is, we use the training data in two different forms and combine them using MTL techniques. The query-document pairs are used for seq2seq tasks. The document is the source sequence and the query is the target sequence. Nogueira et al. [148] also used query-document pairs similarly, but their goal was to predict queries for passage/summary enrichment. We also use the MS-MARCO QA dataset by converting the data into a seq2seq format for MTL training by concatenating the query and the relevant passage as the source sequence and use the human written answer as the target sequence, which is similar to conditioned summarization [65].

Task Conditioning. Task conditioning is a crucial component in achieving effective MTL. The approach we take is similar to T5 [164], where special task identifiers are added at the beginning of input sequences. For example, *rank:* is added for ranking tasks. The input for ranking becomes: *rank: <passage> <query>*. In this work, we use the identifier "*rank:*" for ranking, "*sum:*" for query generation, and "*answer:*" for question-answering. The specific text for task conditioning is arbitrary as long as it is unique for each task. During training, two or three heads are used depending on the number of generative tasks used as input. For inference, we ignore the seq2seq heads and use only the ranking output layer.

5.4.3 Training Method

One training instance contains one query, one positive document and one negative document (and one answer for the query if we use the QA task). We summarize in Table 5.3 how we feed the three components into the models. The principles applied are intuitive. For encoders sequences need to be concatenated while for encoder-decoders there are two ways to feed queries and passages for ranking, separately or concatenated, as the names GDMTL_S and GDMTL_C suggest.

Table 5.3: Model inputs. P represents passage; Q represents query; A represents answer; ⊕ represents concatenation. Encoders require the sequences to be concatenated. Encoder-decoders have two ways to feed queries and passages for ranking, separately or concatenated.

	Ranking Input		Generative Input	
	Encoder	Decoder	Encoder	Decoder
mono	P⊕Q	-	-	-
BERT	P⊕Q	-	-	-
GDMTL	P⊕Q	-	P⊕Q	-
GDMTL+	P⊕Q	-	P⊕Q or P⊕Q⊕A	-
BART_S	P	Q	-	-
GDMTL_S	P	Q	P	Q
BART_C	P⊕Q	P⊕Q	-	-
GDMTL_C	P⊕Q	P⊕Q	P	Q

Using GDMTL as an example, we first use the concatenation of the document and the query as input and calculate ranking loss using Eq 5.3. Then we use the concatenation of the positive document-query pair only and instruct the model to use mixed attention and calculate the generative task loss using Eq 5.4. The two losses are weighted according to their uncertainty and summed up for back-propagation.

5.4.4 Model Implementation

In this experiment, we use BERT as the encoder-only architecture and BART as the encoder-decoder architecture, but our approach is easily amenable to any similar transformer architecture such as the ones made available by Hugging Face.[3] We implement a single task learning (STL) baseline and all MTL approaches using Transformers[4] and PyTorch library. All models are trained using mixed-precision floating point arithmetic on two NVIDIA V100-32GB GPUs. We use the AdamW [122] optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and weight_decay $= 5e^{-5}$ for training. We have observed that training works best when using small learning rates (the name **fine**tuning is a clue), and have set our learning rate accordingly to $2e^{-5}$. We use a linear scheduler to adjust the learning rate step-wise with a one epoch warm-up and decays linearly to a minimum value of $1e^{-6}$. All models are trained for 10 epochs, and final epoch selection is a hyper-parameter decided during cross-validation.

For ranking tasks, the length of the input sequence is limited to 256 tokens. For seq2seq tasks, the length of the encoder input is limited to 256 tokens and the decoder input to 56 tokens. We do not use a maximum length of 512 tokens as the default, since the average length of the MS-MARCO training passages is 91 terms and the average length of the training queries are around 6 terms. Setting the maximum length greater than 256 results in no measurable differences in effectiveness and increases training costs, so we have limited it. Due to data alignment constraints in GPUs, token padding tends to lower computational throughout but is ultimately discarded at aggregation time when using short input sequences. For other details, see the code repository for this work which is publicly available. [5]

Baseline Performance. Before diving into the results, we present a baseline implementation BERT using monoBERT [145, 147] as a reference since they are conceptually similar. We apply two-stage retrieval in our experiments: BM25 as the first stage ranker to retrieve top 1,000 passages, and then neural ranking models as the second stage ranker for re-ranking. Note that we use the base version of both BERT and BART, which have $110M$ and $139M$ parameters respectively, while monoBERT is derived using a"large" BERT model which contains $340M$ parameters.

Our new implementation of BERT significantly outperforms monoBERT for all effectiveness metrics we tested at the $p < 0.01$ level, which is shown in Table 5.4. There are several implementation differences between BERT and monoBERT, and we are unable to attribute the effectiveness

[3] https://huggingface.co
[4] https://github.com/huggingface/transformers
[5] https://github.com/binshengliu/gdmtl

differences to any one of these. Our best conjecture is that we have trained our model using a pair-wise loss while monoBERT used a point-wise. This outcome does not agree with the previous findings of Han et al. [78] who found only small difference between the two approaches for the MS-MARCO collection. Exploring the differences further is beyond the scope of this work as it is orthogonal to our primary aims. Our code is available if anyone wishes to explore it further.

These preliminary results show that competitive baselines are being used to benchmark our new approaches.

5.5 Results and Analysis

In this section, we attempt resolve our original research questions. In Sec 5.5.1 we test if our approach improves retrieval effectiveness on two different test sets and provide failure analysis. In Sec 5.5.2 we test if our approach can generalize well with different training and testing distributions. Then in Sec 5.5.3, we provide some other related analysis. Finally, in Sec 5.5.4 we discuss the impact of implementing MTL when using different architectures.

5.5.1 Improving Ranking Effectiveness

RQ1. Do generative tasks improve discriminative retrieval models?

We first answer RQ1 by experimenting using MS-MARCO and CAsT 2019 test queries, each of which have different properties. MS-MARCO contains thousands of queries with shallow judgments while CAsT 2019 has fewer queries and deep judgments. MS-MARCO also contains several related tasks such as Question-Answering and Query Categorization in addition to the ranking task. Results using query generation and the QA tasks as the auxiliary tasks are provided for MS-MARCO, and include a further analysis using query categorization.

Table 5.4: Retrieval performance of STL and MTL models on MS-MARCO. $^{\triangle}$ and $^{\blacktriangle}$ indicate statistical significance at $p < 0.05$ and $p < 0.01$ over BERT with Holm-Bonferroni correction, respectively.

	MRR		NDCG		RBP	
	10	100	10	20	0.5	0.8
BERT	0.384	0.393	0.448	0.471	0.177 +0.823	0.099 +0.901
GDMTL	0.392$^{\blacktriangle}$	0.401$^{\blacktriangle}$	0.454$^{\triangle}$	0.476$^{\triangle}$	0.182 +0.818$^{\blacktriangle}$	0.101 +0.899$^{\triangle}$
GDMTL+	0.394$^{\blacktriangle}$	0.403$^{\blacktriangle}$	0.458$^{\blacktriangle}$	0.480$^{\blacktriangle}$	0.182 +0.818$^{\blacktriangle}$	0.101 +0.899$^{\blacktriangle}$
BM25	0.244$^{\blacktriangledown}$	0.256$^{\blacktriangledown}$	0.299$^{\blacktriangledown}$	0.324$^{\blacktriangledown}$	0.108 +0.892$^{\blacktriangledown}$	0.067 +0.933$^{\blacktriangledown}$
mono	0.372$^{\blacktriangledown}$	0.381$^{\blacktriangledown}$	0.433$^{\blacktriangledown}$	0.455$^{\blacktriangledown}$	0.172 +0.828$^{\blacktriangledown}$	0.096 +0.904$^{\blacktriangledown}$

MS-MARCO. We begin our analysis with a head-to-head comparison of single task and multi-task learning as described in the previous section using MS-MARCO. Table 5.4 summarizes the

performance comparisons when incorporating the query generation task into GDMTL. We found that the GDMTL and GDMTL+ models significantly outperform their STL counterpart BERT for every metric, showing that the addition of the generative task can improve the performance of a discriminative ranking model. In the next section, we will more exhaustively analyze *how* different generative tasks reinforce different aspects in the discriminative ranking model task, which translated to the improved performance we observe here.

When we consider the addition of a third task, such as the QA task, GDMTL+ shows even more improvement across all the effectiveness metrics tested. The QA dataset reinforces the model by providing additional information that connect queries and the most important part of the passage. Consider the example query "what county is columbus city in", the passage "Columbus is a city in [...] Bartholomew County [...]. The population was [...]" which is very general description, and the answer "Columbus is a city in Bartholomew County." The final answer focuses the attention on the most important words in the passage and thus further strengthens contextual information in the model.

Table 5.5: Pair-wise win/tie/loss analysis on MS-MARCO dev set based on MRR@10, indicating the number of queries being improved, unchanged (within 10%), and hurt. The comparing base is listed in the headers.

	mono			BERT			GDMTL		
	W	T	L	W	T	L	W	T	L
BERT	1735	3775	1470		-			-	
GDMTL	1755	3805	1420	1375	4333	1272		-	
GDMTL+	1827	3764	1389	1445	4281	1254	1317	4459	1204

In Table 5.5 we show the MRR@10 comparison using monoBERT, BERT, GDMTL, and GDMTL+ as a pairwise win/tie/loss comparison. The trend is consistent across all the comparisons. From top to bottom, more queries get improved than are degraded as more tasks are added. From left to right, it is also clear that when the base system is more competitive, it becomes harder to get any large improvements.

We also analyze all the document ranking changes between BERT and MTL systems in Figure 5.7. Here, GDMTL and GDMTL+ rank 71 and 74 more documents respectively at position 1 than BERT. Note that BERT already achieves a perfect score for 26% of all queries, with nearly 50% of all queries retrieving a relevant passage in the top 3 positions, which means in practical terms that dramatic improvements for BERT are not possible as there are few opportunities to achieve big gains. Nevertheless, it is clear that GDMTL and GDMTL+ do in fact consistently rank relevant documents at higher positions than BERT alone.

Caveats. We must also note that the residuals shown in Table 5.4 are unacceptably high, even when using RBP, $p = 0.5$ which is analogous to MRR. This suggests that many of the results returned in higher positions for each query are unjudged, and we can therefore not be sure

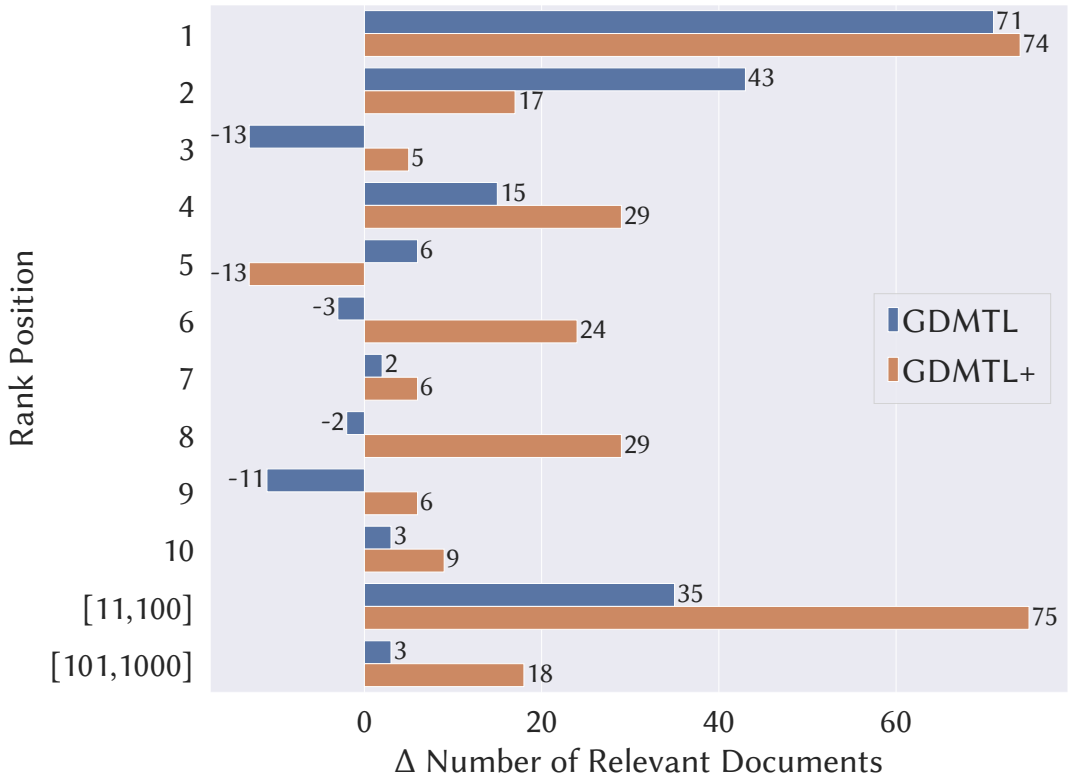


Figure 5.7: Rank position changes for relevant documents between BERT and MTL systems. Each bar on the right represents more documents added to a rank position. Conversely, bars on the left indicate that more documents moved from this position to another than were added. The more documents added to higher positions, the higher the overall effectiveness.

they are not relevant. As there is generally a single positive judgment provided for each query, this should not be surprising, and of course applies to all experimental studies appearing in the literature using the MS-MARCO collection. Nevertheless, residuals of this magnitude dictate that fine-grained comparisons must be interpreted with caution. In this work, we are most interested in comparing relative trends between closely related models in order to understand how multi-task learning can be used to improve model quality, and not in achieving the most competitive result for any particular collection, and is therefore sufficient for our current needs. This is an important problem that does warrant further analysis in the future.

Per-Category Breakdown Analysis. MS-MARCO also provides query categorization, so we can analyze where MTL exhibits the most benefit. Table 5.6 shows that Numeric, Location and Person queries tend to be highly effective with no MTL. These queries are relatively straightfor-

ward and can usually be resolved using the keywords. Thus, they do not rely heavily on semantic understanding. Results for Entity queries are somewhat surprising. Therefore, we manually inspected a few of these queries. This qualitative analysis suggests that some answer passages are not as easy to identify as one might think. For example, the entity query "what can help dogs sleep" also implies that passages on "why" dogs cannot sleep are also relevant. In this case, the answer preferred by the assessor were about "aches and pains" in dogs, with "aspirin" being the remedy proposed, with sleep problems being a symptom of the condition being discussed in the passage. Another entity query "highest dosage of aspirin" also cannot be simply answered with one keyword. The answer also depends on contextual information such as age and any related health conditions. Ultimately, a single passage selected by the original assessor is known and why it was ultimately selected is open to interpretation, and requires caution, as there are clearly other passages being returned that may or may not also be relevant, as alluded to by the RBP residuals discussed above. It is clear the Entity queries seem to be more difficult for this collection. The last type of query is a Description query. These queries often require longer answers and thus have the highest demand of semantic learning, for example "what does the chief administrator do". Description queries benefit the most when using MTL, and is in line with our expectations.

Table 5.6: MRR@10 of STL and MTL models on MS-MARCO by query type. $^{\triangle}$ and $^{\blacktriangle}$ indicate statistical significance at $p < 0.05$ and $p < 0.01$ over BERT with Holm-Bonferroni correction.

	Description (53.12%)	Numeric (26.12%)	Entity (8.81%)	Location (6.17%)	Person (5.78%)
BERT	0.369	0.391	0.344	0.473	0.443
GDMTL	0.377	0.395	0.361	0.489	0.441
GDMTL+	0.383$^{\triangle}$	0.395	0.351	0.489	0.439

Enriching the model with query generation and question answering tasks strengthens semantic and contextual dependencies in the model. Small perturbations when training the model improve performance for complex queries.

CAsT 2019. Table 5.7 provides a summary level effectiveness comparison when applying our models on the new test collection, and more detailed the win/tie/loss analysis for CAsT 2019 is shown in Table 5.8. Note is not a corresponding QA dataset for CAsT 2019 queries, so we could not test the GDMTL+ model. Nevertheless, the overall trend is consistent in MS-MARCO albeit with larger margins. Since CAsT 2019 includes graded relevance judgments, the rank position comparison shown for MS-MARCO may be less informative as measures such as NDCG tend to combine rank, grades, and gain functions such that the score is an aggregate of every document shift, and not just the highest ranking one. So, we have plotted the per query differences for NDCG@10 when using BERT and GDMTL instead, which is shown in Figure 5.8. This figure shows that the total number of wins with GDMTL is higher, and each of the differences tends to be larger on average.

Table 5.7: Retrieval performance of STL and MTL models on CAsT 2019. ▵ and ▲ indicate statistical significance with $p < 0.05$ and $p < 0.01$ over corresponding BERT with Holm-Bonferroni correction.

	MRR		NDCG		RBP	
	10	100	10	20	0.5	0.8
BERT	0.544	0.550	0.405	0.428	0.366 +0.286	0.317 +0.355
GDMTL	0.590▵	0.595▵	0.431▵	0.451▵	0.419 +0.236▲	0.341 +0.330▵
BM25	0.474▽	0.479▽	0.331▼	0.346▼	0.331 +0.259	0.284 +0.361▽

Table 5.8: Pair-wise win/tie/loss analysis on CAsT 2019 based on NDCG@10, indicating the number of queries being improved, unchanged (within 10%), and hurt.

	BM25			BERT		
	W	T	L	W	T	L
BERT	87	39	47	-		
GDMTL	93	41	39	65	59	49

5.5.2 Improving Model Generalizability

RQ3. Are the resulting models easily transferable to other retrieval tasks?

In this section, we discuss preliminary results on task-based transfer learning. We use our model trained on MS-MARCO and the queries from the CAsT 2019 collection, as shown in Table 5.9.

Table 5.9: Retrieval performance of transferring models from MS-MARCO to CAsT 2019. ▵ and ▲ indicate statistical significance with $p < 0.05$ and $p < 0.01$ over BERT with Holm-Bonferroni correction.

	MRR		NDCG		RBP	
	10	100	10	20	0.5	0.8
BERT	0.617	0.617	0.486	0.490	0.470 +0.230	0.392 +0.318
GDMTL	0.656	0.651	0.495	0.510	0.483 +0.231	0.402 +0.307
GDMTL+	0.686▵	0.683▵	0.498	0.498	0.514 +0.202▵	0.409 +0.301
BM25	0.474▼	0.479▼	0.331▼	0.346▼	0.331 +0.259▼	0.284 +0.361▼

Again, care must be taken when interpreting these results. As shown in Table 5.10, queries share a common information need and have overlapping qrels. A similar observation can be made in other tasks that use the MS-MARCO collection, and is a point of discussion with the

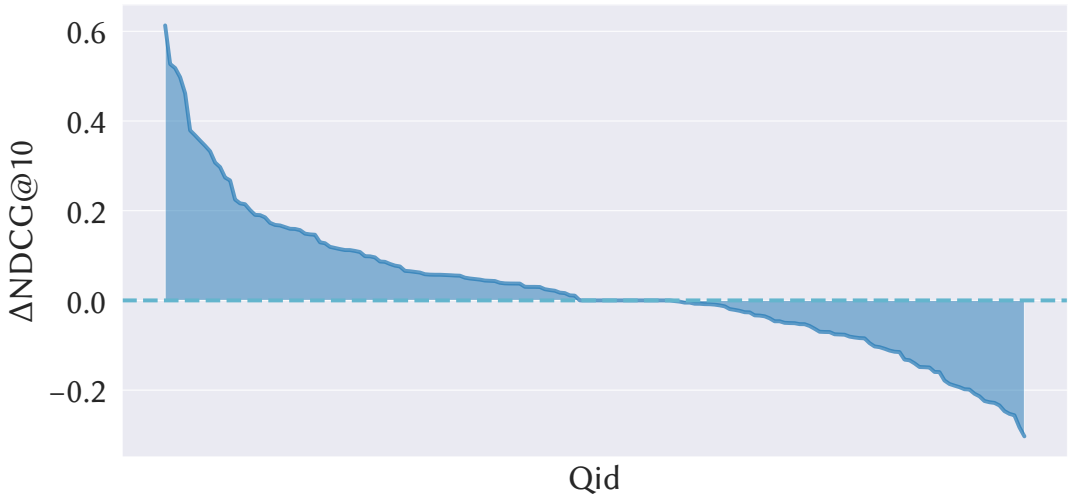


Figure 5.8: Per-query breakdown of NDCG@10 on CAsT 2019. Queries in descending order of score difference.

organizers. The collection was created and maintained by Microsoft, and in live production environments, duplicates and near duplicates are common, especially in very large training sets such as MS-MARCO. At any rate, the performance of the transfer models seem to be superior to models that are trained independently for CAsT 2019. That is, all models tested showed similar advantages, and we are most interested in a relative comparison here. When compared head-to-head in identical scenarios, GDMTL and GDMTL+ are consistently more effective than BERT.

In Table 5.9, consistent improvements are observed but are only significant for MRR@10, MRR@100 and RBP $p = 0.5$ for GDMTL+. The lack of significance for deeper metrics may be an artifact of the shallow judgment pool, the near replicates in training data, or both. There are two important factors to account for this phenomenon. First, 90% of the MS-MARCO training data has only one relevant passage for training, while the CAsT 2019 queries have 17 relevant passages on average. Second, the MS-MARCO judgments are binary, but the CAsT 2019 judgments are graded. Such discrepancies may also contribute to a reduced overall performance when directly transferring a model to a new task without any modification. We will explore this further in future work.

5.5.3 Additional Result Analysis

Impact of Training Data Size. We now turn our attention to understanding what impact the number of training instances has in our MTL model. In this analysis, we test our approaches using samples of 75%, 50%, 25%, and 12.5% of the original MS-MARCO training data, while fixing all other hyperparameters. In order to minimize noise from sampling, we apply top down subset

Table 5.10: CAsT 2019 evaluation queries and MS-MARCO training queries share some information needs.

Collection	Topic	Rel Doc
CAsT 2019	69_6: What are the side effects of melatonin?	97921
MS-MARCO	564795: what are side effects of melatonin pills	97921
CAsT 2019	67_8: What are anemia's possible causes?	883439
MS-MARCO	556252: what are causes of very bad anemia	883439
CAsT 2019	31_7: What is the first sign of throat cancer?	7035854
MS-MARCO	574369: what are the symptoms of throat cancer	7035854

sampling. For example, we take 75% from the original set, and then take 66.7% from the 75% subset to create the 50% set and so on. We do not run the same experiment using CAsT 2019 as the total number of training instances is too small.

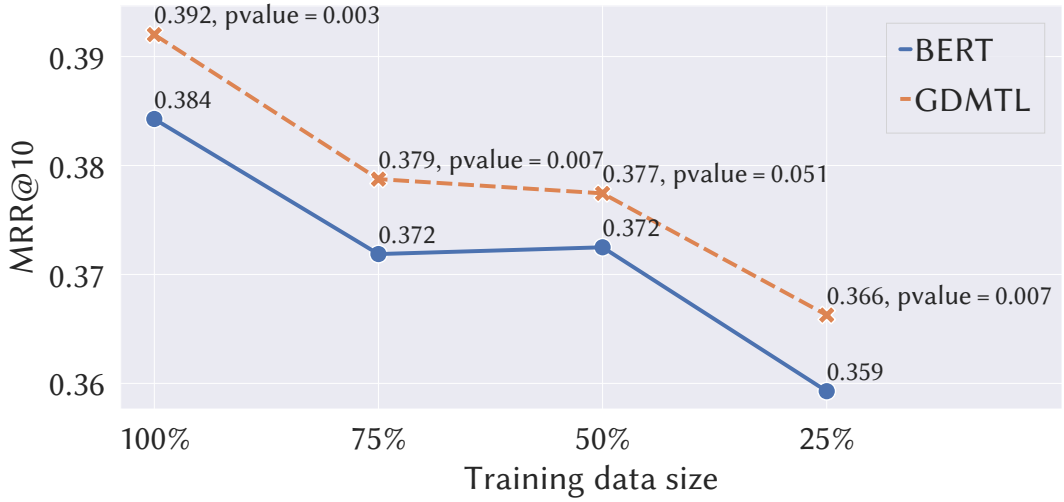
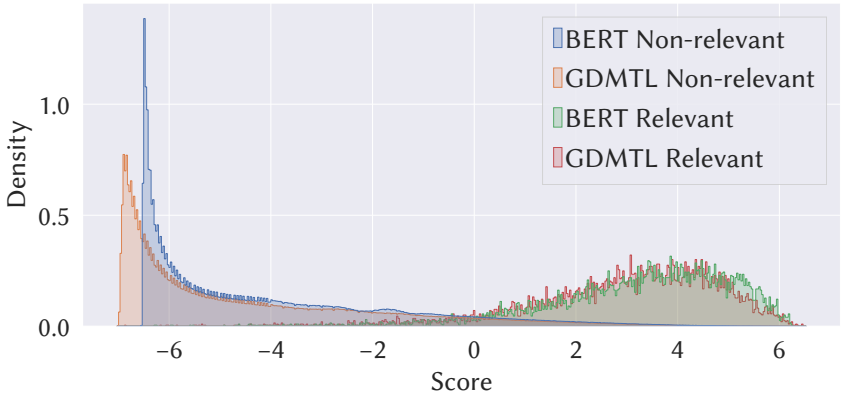


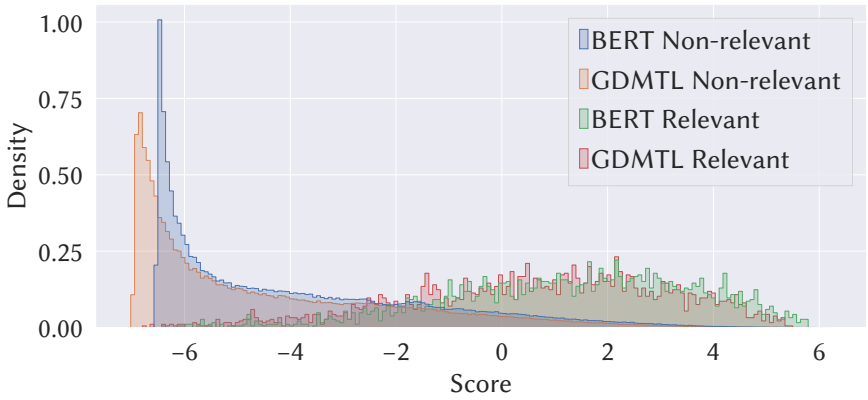
Figure 5.9: MTL consistently improves the model as the number of training instances is increased.

Results are shown in Figure 5.9. Regardless of training set size, GDMTL consistently outperforms BERT. The improvements are significant except when a 50% sample of training data was used. So, our initial experiments suggest that MTL can be benefit even in cases where the training data is limited.

Score Distributions. In term-based retrieval systems, raw retrieval scores across queries are often not comparable as they depend on query length. For example, query likelihood models assign a score to a query-document pair $P(Q|D) \propto \prod_{q \in Q} P(q|D)$ where q is a term in query Q and D is a document. Longer queries produce lower scores. However, attention-based neural ranking models apply normalization at attention aggregation time. Before the final prediction, the query and the document are projected into sentence representations in a latent space, for example [CLS] of BERT. The contribution of each token to the sentence representation is softmax-normalized, so the different query lengths do not change the distribution of the prediction scores.



(a) Score distributions for MS-MARCO.



(b) Score distributions for CAsT 2019.

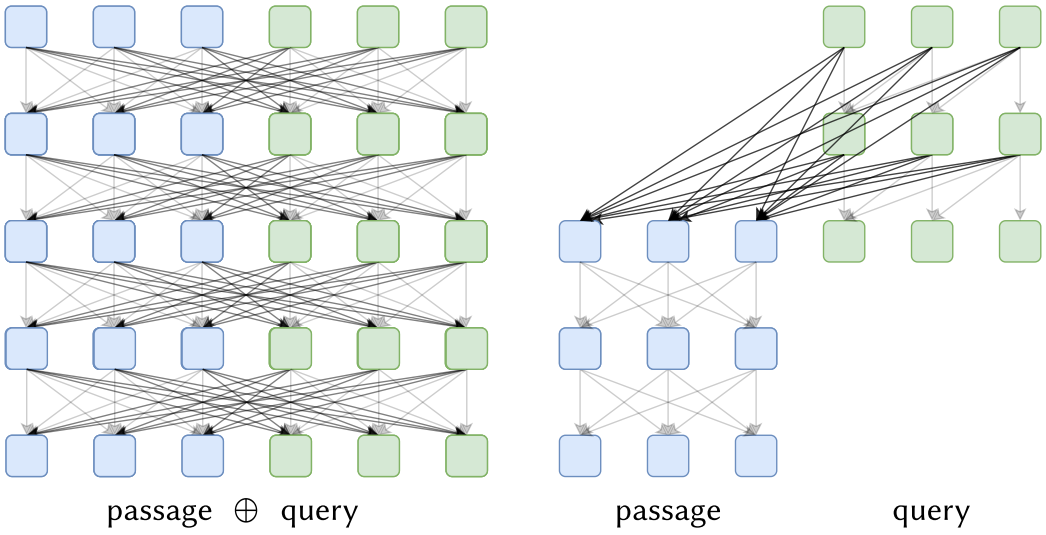
Figure 5.10: Score distributions for MS-MARCO and CAsT 2019.

A plot of the histogram of scores produced using our models is shown in Figure 5.10. Relevant documents are mainly on the right and non-relevant documents are on the left. More importantly,

the figure indicates how MTL has made the models more discriminative by assigning lower scores to non-relevant documents.

5.5.4 Impact of Architectures

RQ2. Which neural network architectures (i.e., encoder-only or encoder-decoder) produce more effective joint discriminative and generative relevance models?



(a) Encoder attention. **(b)** Encoder-decoder attention.

Figure 5.11: Encoder and encoder-decoder models require different interaction mapping (lines between different colors) when including passage and query pairing.

As we discussed in Section 5.3, not only the task balance strategy can affect the performance of an MTL framework, but also the underlying model architecture. In this experiment, we focus on two commonly used architectures for tasks: encoder-only (BERT) and encoder-decoder (BART). For the STL models, BERT has higher retrieval effectiveness than BART_S. There are two plausible reasons for this difference. First, BERT and BART are pretrained differently; second the encoder-decoder architecture must separate query-document interactions across the two layers as illustrated in Figure 5.11. In order to determine which of these contribute the most to the performance differences we have observed, we attempted to increase query-document interaction signals by feeding the concatenation of query and document into the BART encoder and decoder (shown as BART_C in Table 5.11). As we can see, inducing a stronger query document interaction results in improved performance when compared with BART_S, where interactions occur primarily in the decoder.

When including auxiliary query generation tasks, we can observe that GDMTL_S can outperform BART_S significantly, but GDMTL_C also improves BART_C, but not as significantly. This

Table 5.11: Retrieval performance of BART-based STL and MTL models on MS-MARCO. BERT and GDMTL are listed as references. ▵ and ▲ indicate statistical significance with $p < 0.05$ and $p < 0.01$ over BART_S with Holm-Bonferroni correction.

	MRR		NDCG		RBP	
	10	100	10	20	0.5	0.8
BART_S	0.370	0.380	0.436	0.460	0.170 +0.830	0.097 +0.903
GDMTL_S	0.382▲	0.392▲	0.445▲	0.470▲	0.176 +0.824▲	0.099 +0.901▲
BART_C	0.385▲	0.394▲	0.451▲	0.474▲	0.178 +0.822▲	0.100 +0.900▲
GDMTL_C	0.390▲	0.399▲	0.454▲	0.478▲	0.180 +0.820▲	0.101 +0.899▲
BERT	0.384▲	0.393▲	0.448▲	0.471▲	0.177 +0.823▲	0.099 +0.901▲
GDMTL	0.392▲	0.401▲	0.454▲	0.476▲	0.182 +0.818▲	0.101 +0.899▲

is possibly caused by the different shape alignments required in the two tasks, as shown in Table 5.2. We also observed that both GDMTL (corrected $p = 0.0205$, 95% confidence interval $[0.0038, 0.0171]$) and GDMTL_C (corrected $p = 0.0260$, 95% confidence interval $[0.0029, 0.0141]$) also outperform GDMTL_S significantly for MRR@10 when compared directly, showing architecture choices do have an important role when designing MTL models.

To summarize, BERT and BART_C achieve similar performance when used only for ranking. Both architectures benefit from incorporating generative tasks but GDMTL achieves the best retrieval effectiveness in our experimental setup. Using mixed attention, GDMTL is able to incorporate two heterogeneous tasks using unified inputs. In contrast, GDMTL_S is also able to combine multiple tasks in one model, but can suffer from the reduced query-document interactions due to the architectural requirements. GDMTL_C leverages additional query-document interactions, but input shapes are more likely to be task specific. However, encoder-decoder transformers (GDMTL_S, GDMTL_C) in general may be less suitable than encoder transformers (GDMTL) if the primary goal is only ranking effectiveness, but may be a better choice for generation tasks, or if your overall goal is to use all the task heads, and not just one.

5.5.5 Generation Quality

We have seen evidence that the query generation task has positive impacts on ranking effectiveness. Now we examine how the ranking task impacts the effectiveness of query generation. To perform the analysis, a new model BART_G was trained with only the query generation task. The GDMTL_S model we introduced previously was trained with both the ranking and query generation tasks. Now we denote it as GDMTL_G since we focus on its generation capability.

We randomly sample 1000 query-document pairs from the dev set for the comparison. We use BART_G and GDMTL_S to generate 1000 queries respectively given the sampled documents. These queries are used to re-rank the corresponding retrieval lists in the dev set. The results

Table 5.12: Retrieval effectiveness of using generated queries for retrieval. No significant difference is observed.

	MRR		NDCG		RBP	
	10	100	10	20	0.5	0.8
BART_G	0.486	0.494	0.549	0.568	0.226 +0.774	0.118 +0.882
GDMTL_G	0.480	0.489	0.543	0.562	0.224 +0.776	0.118 +0.882

Table 5.13: Example queries generated by BART_G and GDMTL_G.

	Query
Original	heart rate increase during exercise
BART_G	why does your heart rate increase during exercise
GDMTL_G	how does heart rate increase during exercise
Original	does body fat weigh more than muscle
BART_G	effects of too much body fat
GDMTL_G	how much body fat can you get
Original	how long do dissolvable stitches take
BART_G	how long for stitches to dissolve
GDMTL_G	how long to dissolve stitches after surgery

are shown in Table 5.12. No significant difference is observed according to the table. A win-tie-loss analysis shows that GDMTL_G wins 194 queries, ties 604 queries, and loses 202 queries. However, the ranking task still shows impact on the final generation. Table 5.13 shows several example queries. Overall, 978 and 984 queries are different from the original queries for BART_G and GDMTL_G respectively. Between BART_G and GDMTL_G, 719 queries are different. Further examining the significance of the difference is an interesting direction.

5.6 Summary

In this chapter, we looked at query generation from a new point of view. We investigated methods of incorporating relevance signals in query generation into a ranking model. The relevance signal contained in query generation can be observed from the Language Modeling retrieval model where the generation probability is used for document ranking. The idea of combining generation and ranking also builds on our current knowledge about probabilistic models in IR: generative models and discriminative models. We have proposed the GDMTL framework, which integrates generative and discriminative tasks via multi-task learning. Our framework exploits readily available generative tasks such as query generation and QA tasks to improve discriminative retrieval models. Our experiments have answered RQ1 affirmatively – generative tasks

are indeed able to improve the performance of discriminative retrieval models. Regarding RQ2, we have provided detailed comparisons between architectures which uncover additional insights that allow us to better understand how MTL model differences are affecting performance. Finally, for RQ3 we show that the models learned using our approach are easily transferable, which is beneficial in new tasks where little training data may be available.

Several interesting open questions remain. First, we do not fully understand how generative tasks modify model behaviors via attention and gradient-based analysis. Understanding what attention learns is still an active research area. Some recent work has presented methods to analyze the semantic meanings of attention [35]. Gradient-based analysis [193] explores how each word contribute to the final prediction. These methods could possibly shed light on how query generation signals are transferred into a ranking model. Second, we haven't explored extending our method to long documents. Using Transformers on long documents is mainly limited by Attention's quadratic time and space complexity. Beltagy et al. [15], Hofstätter et al. [82] both use smaller windows to reduce the computational cost. This also leads to another interesting aspect: efficiency. Khattab and Zaharia [98] delay the interactions between a query and a document to improve efficiency. Improving Transformer efficiency is an open and challenging problem of great practical importance.

6 Summary

Queries are the key to understanding a user's information need, but they are surrounded by may contain various issues impeding retrieval effectiveness such as spelling errors, vague or ambiguous representations, and under-specified information needs. To better represent a user's information need, query optimization is often the tool of choice.

In this thesis, we first study extending a traditional query optimization technique: relevance modeling in unstructured documents and structured documents. Web pages, product pages, job listings, etc. all contain structural information which implies term importance. By leveraging this structural information, we propose a field-based relevance modeling technique which is able to induce relevance models from document fields, and a method to apply relevance models on document fields. Our results show that retrieval effectiveness can be improved with this addition information. However, relevance modeling of queries has obvious shortcomings. They are often inefficient, uninterpretable, and tend to make small improvements for web documents.

We then conduct a user study to explore the value of completely rewriting a query. The analysis is performed using human-written query variations and automatically generated queries from a commercial search engine. Importantly, our analysis first shows that relevance models can hardly achieve the same level of improvements as rewriting queries, indicating that remarkable effectiveness gains are possible purely based on query rewriting. Both automatic techniques and humans can generate effective queries, but effectiveness differences still exist. We need more powerful generation models to achieve human-level quality.

To generate more effective queries, we then inspect leveraging state-of-the-art transformer models for query generation which is a fundamental technique that can be used in various query optimizations. Now we combine two important objectives: effectiveness from the IR community and readability from the NLP community. Effectiveness is a core objective in IR while readability is an important metric in query interpretability and transparency. As neural language processing advances, we are better positioned to explore the readability of queries. We propose a novel task – *strong natural language query* which aims at generating effective natural language queries. Our solution consists of a supervised learning stage for readability training and a reinforcement learning stage for effectiveness training. First, readability is improved using abstractive summarization data. However, a good summary is not necessarily a good query. So, we then use reinforcement

learning to directly optimize for effectiveness metrics which are often non-differentiable. The combined approach leads to the best trade-off between the two dimensions.

Finally, as natural language queries are common due to the increased popularity of voice search and digital assistants, we explore jointly modeling ranking and query generation in a multi-task learning framework to improve ranking effectiveness. The core idea is that understanding natural language queries and generating natural language queries share common knowledge thus a joint modeling approach may benefit both sides. In the long history of IR, direct ranking models such as an SVM and generation-based models such as query likelihood both showed promising results in capturing relevance signals. Our first contribution is a framework which combines both through multi-task learning techniques where knowledge is shared by common parameters. Our second contribution is to implement the framework using transformers. Combining ranking and query generation is difficult as they have different requirements on the model architectures. First, we use an attention-based approach to incorporate query generation into an encoder-only architecture. Second, we propose a method for using an encoder-decoder for ranking. We are able to carry out joint modeling using either architecture. Our third contribution is an in-depth failure analysis on the impact of transformer architectures on capturing relevance signals.

6.1 Future Directions

Understanding *Ranking* in Transformers. Transformers often contain many layers that are responsible for resolving different linguistic features. Clark et al. [35] used a visualization and probing methods to understand how linguistic features are captured inside a transformer. They found certain layers identify synonyms, certain layers resolve pronouns, certain layers detect object of verbs (shown in Figure 6.1), certain layers are responsible for verbs, and so forth. While these generalized linguistic knowledge is fascinating, how attention weights affect the final relevance score is not clear at all. *Is there a layer or a head that particularly attends to words that are important for estimating ranking scores? Does multi-task learning (e.g. GDMTL) change attention weights to achieve higher effectiveness?* Ablating transformer layers and heads to analyze the impact on the output scores may help answer these questions. A better understanding to transformers can also help us prune less-important layers so we can reduce model size and improve efficiency.

Using attention weights to understand a decision-process is very attractive due to its simplicity and similarity to human attention. However, researchers have shown that attention weights may not directly correlate to the final output score [87]. Gradient-based methods [193] are perhaps more promising for such an analysis. For example, they can be used to break down the relevance score to every single word in a document. A relevance score of 0.987 may be composed of 0.3 from word A in the query, 0.2 from word B in the document, etc. It would be of great value to leverage NLP tools to understand transformers' decision-making process with a focus on IR.

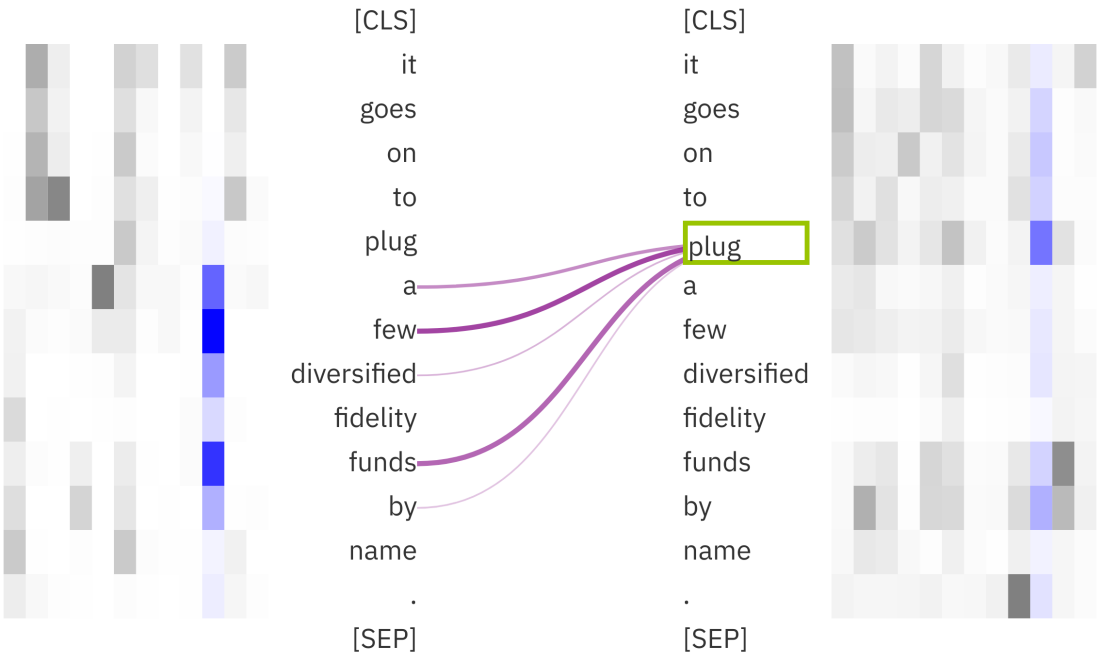


Figure 6.1: An attention layer responsible for resolving verb-object. Image produced from https://huggingface.co/exbert/?model=bert-base-uncased.

Neural Relevance Models. With query generation models, there are many possible applications. One of them is to generate queries for pseudo-relevant documents and use the new queries to improve the original query. This approach has a strong connection to the classic relevance models proposed by Lavrenko and Croft [106] so we refer to the prospective method as *neural relevance models* – relevance models estimated directly using a generative model instead of statistical language models. One direction is to use a neural language model which does not take into account word ordering. This approach fits nicely into the original relevance model framework. That is, $P(w|D)$ in Eq 3.3 is now estimated with a neural language model. Another direction is to retain word ordering and use a generative model to directly rewrite queries given a list of pseudo-relevance documents.

Optimizing GDMTL for Generation Effectiveness. In Chapter 5 we used multi-task learning to jointly model ranking and generation. The former has an effectiveness objective while the latter has a readability objective. We try to transfer the ranking knowledge to generation, but it is very subtle how this works without a direct optimization. How the generation for the GDMTL framework can be directly optimized for effectiveness is unclear. That is, can we optimize ranking effectiveness using supervised learning while optimizing generation effectiveness using reinforcement learning?

Learning Efficiency. Training a transformer for either ranking or generation is time-consuming. Due to the task conditioning (instructing the model whether it is a ranking or a generation task), the same sequence must go through the network multiple times. Developing a way to reduce this constant cost would increase the training speed by approximately 50%.

Modeling for Documents. The last direction we hope to further explore is to extend our work to documents. Our work mostly focused on passages or short documents which often contain no more than 200 words. This was largely limited by the complexity of the attention mechanism which has a time and space complexity of $O(N^2)$ where N is the length of the input sequence. Applying the original transformer on long documents is thus infeasible practically. Several researchers are trying to solve this problem. In IR, Hofstätter et al. [82] proposed a local window-based attention method that reduced the complexity to $O(N \times l)$ where l is the window size. Mitra et al. [133] proposed changing the order of computing attention weights and reduce the complexity to $O(N \times d)$ where d is the dimension of key vectors and $d \ll N$ for documents. Considering that long documents are pervasive, extending the methods in this thesis to long documents is an interesting and important future direction.

Bibliography

[1] N. Abdul-Jaleel, J. Allan, W. B. Croft, F. Diaz, L. Larkey, X. Li, M. D. Smucker, and C. Wade. UMass at TREC 2004: Novelty and HARD. In *Proc. TREC*, 2004. ⟨57, 61, 62⟩

[2] W. U. Ahmad, K. Chang, and H. Wang. Multi-Task Learning for Document Ranking and Query Suggestion. In *Proc. ICLR*, page 14, 2018. ⟨116⟩

[3] W. U. Ahmad, K. Chang, and H. Wang. Context Attentive Document Ranking and Query Suggestion. In *Proc. SIGIR*, pages 385–394, 2019. ⟨86, 116⟩

[4] L. Azzopardi and M. de Rijke. Automatic Construction of Known-item Finding Test Beds. In *Proc. SIGIR*, page 603, 2006. ⟨84, 86⟩

[5] L. Azzopardi, M. de Rijke, and K. Balog. Building Simulated Queries for Known-Item Topics: An Analysis using Six European Languages. In *Proc. SIGIR*, pages 455–462, 2007. ⟨86, 93⟩

[6] D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proc. ICLR*, 2015. ⟨26, 27, 43, 45⟩

[7] J. Bai, K. Zhou, G. Xue, H. Zha, G. Sun, B. Tseng, Z. Zheng, and Y. Chang. Multi-task learning for learning to rank in web search. In *Proc. CIKM*, pages 1549–1552, 2009. ⟨115⟩

[8] P. Bailey, A. Moffat, F. Scholer, and P. Thomas. User Variability and IR System Evaluation. In *Proc. SIGIR*, pages 625–634, 2015. ⟨14, 15⟩

[9] P. Bailey, A. Moffat, F. Scholer, and P. Thomas. UQV100: A Test Collection with Query Variability. In *Proc. SIGIR*, pages 725–728, 2016. ⟨14, 15, 72, 74, 86⟩

[10] P. Bailey, A. Moffat, F. Scholer, and P. Thomas. Retrieval Consistency in the Presence of Query Variations. In *Proc. SIGIR*, pages 395–404, 2017. ⟨73, 76⟩

[11] J. Beel, B. Gipp, S. Langer, and C. Breitinger. Research-paper recommender systems: A literature survey. *International Journal on Digital Libraries*, 17(4):305–338, 2016. ⟨20⟩

[12] N. J. Belkin. Anomalous states of knowledge as a basis for information retrieval. *Canadian journal of information science*, 5(1):133–143, 1980. ⟨4⟩

[13] N. J. Belkin, C. Cool, W. B. Croft, and J. P. Callan. The effect of multiple query representations on information retrieval system performance. In *Proc. SIGIR*, pages 339–346, 1993. ⟨14⟩

[14] N. J. Belkin, P. Kantor, E. A. Fox, and J. A. Shaw. Combining the evidence of multiple query representations for information retrieval. *Inf. Process. Manag.*, 31(3):431–448, 1995. ⟨14, 15⟩

[15] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The Long-Document Transformer. *arXiv:2004.05150 [cs]*, 2020. ⟨139⟩

[16] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A Neural Probabilistic Language Model. *J. Mach. Learn. Res.*, 3:1137–1155, 2003. ⟨38, 49⟩

[17] R. Benham and J. S. Culpepper. Risk-Reward Trade-offs in Rank Fusion. In *Proc. ADCS*, pages 1:1–1:8, 2017. ⟨72, 73, 76⟩

[18] R. Benham, J. S. Culpepper, L. Gallagher, X. Lu, and J. Mackenzie. Towards Efficient and Effective Query Variant Generation. In *Proc. DESIRES*, pages 62–67, 2018. ⟨15, 57⟩

[19] R. Benham, L. Gallagher, J. Mackenzie, B. Liu, X. Lu, F. Scholer, A. Moffat, and J. S. Culpepper. RMIT at the 2018 TREC CORE Track. In *Proc. TREC*, page 8, 2018. ⟨14, 15⟩

[20] A. Berger and J. Lafferty. Information retrieval as statistical translation. In *Proc. SIGIR*, pages 222–229, 1999. ⟨87⟩

[21] A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra. A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 22(1):39–71, 1996. ⟨113⟩

[22] D. Bertsekas and J. Tsitsiklis. Neuro-dynamic programming: An overview. In *Proc. CDC*, pages 560–564, 1995. ⟨90⟩

[23] B. Billerbeck and J. Zobel. When Query Expansion Fails. In *Proc. SIGIR*, pages 387–388, 2003. ⟨74⟩

[24] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, 3 (Jan):993–1022, 2003. ⟨20⟩

[25] P. Boldi, F. Bonchi, C. Castillo, D. Donato, A. Gionis, and S. Vigna. The query-flow graph: Model and applications. In *Proc. CIKM*, pages 609–618, 2008. ⟨86⟩

[26] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to Rank using Gradient Descent. In *Proc. ICML*, pages 89–96, 2005. ⟨31⟩

[27] C. J. Burges. From RankNet to LambdaRank to LambdaMART: An overview. Technical report, Microsoft, 2010. ⟨26, 110, 113⟩

[28] F. Cai and M. d. Rijke. A Survey of Query Auto Completion in Information Retrieval. *Found. Trends in Inf. Ret.*, 10(4):273–363, 2016. ⟨84⟩

[29] G. Cao, J. Nie, J. Gao, and S. Robertson. Selecting good expansion terms for pseudo-relevance feedback. In *Proc. SIGIR*, pages 243–250, 2008. ⟨62, 63⟩

[30] Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li. Learning to rank: From pairwise approach to listwise approach. In *Proc. ICML*, pages 129–136, 2007. ⟨33⟩

[31] Z. Cao, F. Wei, L. Dong, S. Li, and M. Zhou. Ranking with recursive neural networks and its application to multi-document summarization. In *Proc. AAAI*, pages 2153–2159, 2015. ⟨52⟩

[32] Z. Chen, V. Badrinarayanan, C. Lee, and A. Rabinovich. GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *Proc. ICML*, pages 794–803, 2018. ⟨115⟩

[33] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proc. EMNLP (SSST)*, pages 103–111, 2014. ⟨43⟩

[34] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proc. EMNLP*, pages 1724–1734, 2014. ⟨39⟩

[35] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning. What Does BERT Look at? an Analysis of BERT's Attention. In *Proc. ACL (BlackboxNLP)*, pages 276–286, 2019. ⟨49, 84, 139, 142⟩

[36] C. L. A. Clarke, N. Craswell, and I. Soboroff. Overview of the TREC 2009 Web Track. In *Proc. TREC*, page 9, 2009. ⟨17, 29⟩

[37] C. L. A. Clarke, N. Craswell, and E. M. Voorhees. Overview of the TREC 2012 Web Track. In *Proc. TREC*, page 8, 2012. ⟨17⟩

[38] J. Clarke and M. Lapata. Models for sentence compression: A comparison across domains, training requirements and evaluation measures. In *Proc. COLING*, pages 377–384, 2006. ⟨96⟩

[39] A. Cohan, F. Dernoncourt, D. S. Kim, T. Bui, S. Kim, W. Chang, and N. Goharian. A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents. In *Proc. NAACL-HLT*, pages 615–621, 2018. ⟨87⟩

[40] K. Collins-Thompson and J. Callan. Estimation and use of uncertainty in pseudo-relevance feedback. In *Proc. SIGIR*, page 303, 2007. ⟨62, 63⟩

[41] J. Cordonnier, A. Loukas, and M. Jaggi. On the Relationship between Self-Attention and Convolutional Layers. In *Proc. ICLR*, 2019. ⟨40⟩

[42] M. Crane, J. S. Culpepper, J. Lin, J. Mackenzie, and A. Trotman. A Comparison of Document-at-a-Time and Score-at-a-Time Query Evaluation. In *Proc. WSDM*, pages 201–210, 2017. ⟨96⟩

[43] N. Craswell and M. Szummer. Random walks on the click graph. In *Proc. SIGIR*, page 239, 2007. ⟨5, 15, 73⟩

[44] N. Craswell, B. Mitra, E. Yilmaz, D. Campos, and E. M. Voorhees. Overview of the TREC 2019 deep learning track. In *Proc. TREC*, 2020. ⟨26, 49⟩

[45] N. Craswell, B. Mitra, E. Yilmaz, and D. Campos. Overview of the TREC 2020 deep learning track. In *Proc. TREC*, 2021. ⟨26⟩

[46] W. B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley, Boston, 2010. ISBN 978-0-13-607224-9. ⟨60, 109, 112, 114⟩

[47] S. Cronen-Townsend, Y. Zhou, and W. B. Croft. A framework for selective query expansion. In *Proc. CIKM*, page 236, 2004. ⟨62, 63⟩

[48] S. Cucerzan and E. Brill. Spelling Correction as an Iterative Process that Exploits the Collective Knowledge of Web Users. In *Proc. EMNLP*, pages 293–300, 2004. ⟨3, 12⟩

[49] R. Cummins, M. Lalmas, and C. O'Riordan. The Limits of Retrieval Effectiveness. In *Proc. ECIR*, pages 277–282, 2011. ⟨72⟩

[50] Z. Dai and J. Callan. Context-Aware Sentence/Passage Term Importance Estimation For First Stage Retrieval. *arXiv:1910.10687 [cs]*, 2019. ⟨125⟩

[51] Z. Dai and J. Callan. Deeper Text Understanding for IR with Contextual Neural Language Modeling. In *Proc. SIGIR*, pages 985–988, 2019. ⟨50, 52, 84⟩

[52] Z. Dai, C. Xiong, J. Callan, and Z. Liu. Convolutional Neural Networks for Soft-Matching N-Grams in Ad-hoc Search. In *Proc. WSDM*, pages 126–134, 2018. ⟨52⟩

[53] J. Dalton, L. Dietz, and J. Allan. Entity query feature expansion using knowledge base links. In *Proc. SIGIR*, pages 365–374, 2014. ⟨62⟩

[54] J. Dalton, C. Xiong, and J. Callan. CAsT 2019: The Conversational Assistance Track Overview. In *Proc. TREC*, page 10, 2019. ⟨17, 49⟩

[55] A. P. Dawid and A. M. Skene. Maximum Likelihood Estimation of Observer Error-Rates Using the EM Algorithm. *J. R. Stat. Soc. Ser. C Appl. Stat.*, 28(1):20–28, 1979. ⟨98⟩

[56] M. Dehghani, H. Zamani, A. Severyn, J. Kamps, and W. B. Croft. Neural Ranking Models with Weak Supervision. In *Proc. SIGIR*, pages 65–74, 2017. ⟨110⟩

[57] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. NAACL-HLT*, pages 4171–4186, 2019. ⟨27, 38, 49, 50, 115, 120⟩

[58] F. Diaz. Condensed List Relevance Models. In *Proc. ICTIR*, pages 313–316, 2015. ⟨67⟩

[59] F. Diaz and D. Metzler. Improving the Estimation of Relevance Models Using Large External Corpora. In *Proc. SIGIR*, pages 154–161, 2006. ⟨62⟩

[60] L. Dietz, M. Verma, F. Radlinski, and N. Craswell. TREC Complex Answer Retrieval Overview. In *Proc. TREC*, page 13, 2017. ⟨17⟩

[61] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H. Hon. Unified Language Model Pre-training for Natural Language Understanding and Generation. In *Proc. NeurIPS*, 2019. ⟨87, 93, 94⟩

[62] Y. Du, W. M. Czarnecki, S. M. Jayakumar, R. Pascanu, and B. Lakshminarayanan. Adapting Auxiliary Losses Using Gradient Similarity. *arXiv:1812.02224 [cs, stat]*, 2018. ⟨115⟩

[63] H. Duan and B. Hsu. Online spelling correction for query completion. In *Proc. WWW*, pages 117–126, 2011. ⟨3, 12⟩

[64] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin. Deep Reinforcement Learning in Large Discrete Action Spaces. *arXiv:1512.07679 [cs, stat]*, 2016. ⟨90⟩

[65] A. Fan, D. Grangier, and M. Auli. Controllable Abstractive Summarization. In *Proc. WNMT*, pages 45–54, 2018. ⟨126⟩

[66] K. Filippova and Y. Altun. Overcoming the Lack of Parallel Data in Sentence Compression. In *Proc. EMNLP*, pages 1481–1491, 2013. ⟨96⟩

[67] E. A. Fox. *Extending the Boolean and Vector Space Models of Information Retrieval with P-Norm Queries and Multiple Concept Types.* PhD thesis, 1983. ⟨110⟩

[68] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980. ⟨28, 39⟩

[69] L. Gallagher, R. Chen, J. Mackenzie, F. Scholer, R. Benham, and J. S. Culpepper. RMIT at the NTCIR-13 We Want Web Task. In *Proc. NTCIR*, page 5, 2017. ⟨4, 64⟩

[70] L. Gallagher, R. Chen, R. Blanco, and J. S. Culpepper. Joint Optimization of Cascade Ranking Models. In *Proc. WSDM*, pages 15–23, 2019. ⟨16⟩

[71] L. Gallagher, A. Mallia, J. S. Culpepper, T. Suel, and B. B. Cambazoglu. Feature Extraction for Large-Scale Text Collections. In *Proc. CIKM*, pages 3015–3022, 2020. ⟨26⟩

[72] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional Sequence to Sequence Learning. In *Proc. ICML*, pages 1243–1252, 2017. ⟨39⟩

[73] Y. Goldberg. *Neural Network Methods for Natural Language Processing*. Morgan & Claypool Publishers, San Rafael, Calif., 2017. ISBN 978-1-62705-295-5 978-1-68173-235-0 978-1-62705-298-6. ⟨48⟩

[74] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Proc. NeurIPS*, pages 2672–2680, 2014. ⟨114⟩

[75] J. Guo, Y. Fan, Q. Ai, and W. B. Croft. A Deep Relevance Matching Model for Ad-hoc Retrieval. In *Proc. CIKM*, pages 55–64, 2016. ⟨52⟩

[76] J. Guo, Y. Fan, L. Pang, L. Yang, Q. Ai, H. Zamani, C. Wu, W. B. Croft, and X. Cheng. A Deep Look into neural ranking models for information retrieval. *Inf. Process. Manag.*, 57 (6):102067, 2020. ⟨27, 28, 49, 110⟩

[77] M. Gupta and M. Bendersky. Information Retrieval with Verbose Queries. *Found. Trends in Inf. Ret.*, 9(3-4):209–354, 2015. ⟨6, 100⟩

[78] S. Han, X. Wang, M. Bendersky, and M. Najork. Learning-to-Rank with BERT in TF-Ranking. *arXiv:2004.08476 [cs]*, 2020. ⟨128⟩

[79] T. Hashimoto, H. Zhang, and P. Liang. Unifying Human and Statistical Evaluation for Natural Language Generation. In *Proc. NAACL-HLT*, pages 1689–1701, 2019. ⟨96⟩

[80] C. Hauff, M. Hagen, A. Beyer, and B. Stein. Towards realistic known-item topics for the ClueWeb. In *Proc. IIiX*, pages 274–277, 2012. ⟨86⟩

[81] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, 1997. ⟨28, 39, 41⟩

[82] S. Hofstätter, H. Zamani, B. Mitra, N. Craswell, and A. Hanbury. Local Self-Attention over Long Text for Efficient Document Retrieval. In *Proc. SIGIR*, pages 2021–2024, 2020. ⟨139, 144⟩

[83] Y. Hu, X. Jing, Y. Ko, and J. T. Rayz. Misspelling Correction with Pre-trained Contextual Language Model. *arXiv:2101.03204 [cs]*, 2021. ⟨12⟩

[84] C. Huang, L. Chien, and Y. Oyang. Relevant term suggestion in interactive web search based on contextual information in query session logs. *J. Assoc. Inf. Sci. Technol.*, 54(7): 638–649, 2003. ⟨86⟩

[85] P. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proc. CIKM*, pages 2333–2338, 2013. ⟨21, 52⟩

[86] P. G. Ipeirotis, F. Provost, and J. Wang. Quality management on Amazon Mechanical Turk. In *Proc. HCOMP*, page 64, 2010. ⟨98⟩

[87] S. Jain and B. C. Wallace. Attention is not Explanation. *arXiv:1902.10186 [cs]*, 2019. ⟨142⟩

[88] J. Jiang and W. Wang. RIN: Reformulation Inference Network for Context-Aware Query Suggestion. In *Proc. CIKM*, pages 197–206, 2018. ⟨86⟩

[89] T. Joachims. Optimizing search engines using clickthrough data. In *Proc. KDD*, pages 133–142, 2002. ⟨110, 113⟩

[90] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Evaluating the crowd with confidence. In *Proc. KDD*, pages 686–694, 2013. ⟨98⟩

[91] M. Joglekar, H. Garcia-Molina, and A. Parameswaran. Comprehensive and reliable crowd assessment algorithms. In *Proc. ICDE*, pages 195–206, 2015. ⟨98⟩

[92] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy. SpanBERT: Improving Pre-training by Representing and Predicting Spans. *Trans. Assoc. Comput. Linguistics*, 8(0): 64–77, 2020. ⟨120⟩

[93] K. Kann, S. Rothe, and K. Filippova. Sentence-Level Fluency Evaluation: References Help, But Can Be Spared! In *Proc. CoNLL*, pages 313–323, 2018. ⟨96⟩

[94] S. Karimi, S. Pohl, F. Scholer, L. Cavedon, and J. Zobel. Boolean versus ranked querying for biomedical systematic reviews. *BMC Medical Informatics and Decision Making*, 10(1): 58, 2010. ⟨20⟩

[95] S. Katsumata and M. Komachi. Stronger Baselines for Grammatical Error Correction Using a Pretrained Encoder-Decoder Model. In *Proc. AACL*, pages 827–832, 2020. ⟨12⟩

[96] J. Katzer, J. Tessier, W. Frakes, and P. Das-Gupta. A Study of the Overlap Among Document Representations. In *Proc. SIGIR*, pages 106–114, 1983. ⟨14⟩

[97] A. Kendall, Y. Gal, and R. Cipolla. Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. In *Proc. CVPR*, pages 7482–7491, 2018. ⟨115, 119⟩

[98] O. Khattab and M. Zaharia. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proc. SIGIR*, pages 39–48, 2020. ⟨139⟩

[99] J. Y. Kim and W. B. Croft. A field relevance model for structured document retrieval. In *Proc. ECIR*, pages 97–108, 2012. ⟨64⟩

[100] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *Proc. ICLR*, 2015. ⟨36⟩

[101] J. Koenemann and N. J. Belkin. A case for interaction: A study of interactive information retrieval behavior and effectiveness. In *Proc. CHI*, pages 205–212, 1996. ⟨84, 88⟩

[102] K. Krippendorff. Computing Krippendorff's Alpha-Reliability. *Departmental Papers (ASC)*, 2011. ⟨97⟩

[103] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Proc. NeurIPS*, pages 1097–1105, 2012. ⟨26⟩

[104] R. Kumar, S. Lattanzi, and P. Raghavan. An Algorithmic Treatment of Strong Queries. In *Proc. WSDM*, pages 775–784, 2011. ⟨84, 85, 86, 94, 99⟩

[105] J. Lafferty and C. Zhai. Probabilistic Relevance Models Based on Document and Query Generation. In *Language Modeling for Information Retrieval*, pages 1–10, 2003. ⟨24, 25⟩

[106] V. Lavrenko and W. B. Croft. Relevance Based Language Models. In *Proc. SIGIR*, pages 120–127, 2001. ⟨4, 13, 57, 61, 62, 83, 86, 143⟩

[107] C. Lee and W. B. Croft. Generating Queries from User-Selected Text. In *Proc. IIiX*, pages 100–109, 2012. ⟨86⟩

[108] J. H. Lee. Analyses of Multiple Evidence Combination. In *Proc. SIGIR*, pages 267–276, 1997. ⟨14, 15⟩

[109] K. S. Lee, W. B. Croft, and J. Allan. A Cluster-Based Resampling Method for Pseudo-Relevance Feedback. In *Proc. SIGIR*, pages 235–242, 2008. ⟨62, 63⟩

[110] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, USA, 2014. ISBN 978-1-107-07723-2. ⟨20⟩

[111] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proc. ACL*, pages 7871–7880, 2020. ⟨115, 124⟩

[112] C. Li, Y. Sun, B. He, L. Wang, K. Hui, A. Yates, L. Sun, and J. Xu. NPRF: A Neural Pseudo Relevance Feedback Framework for Ad-hoc Information Retrieval. In *Proc. EMNLP*, 2018. ⟨62, 63⟩

[113] L. Liebel and M. Körner. Auxiliary Tasks in Multi-task Learning. *arXiv:1805.06334 [cs]*, 2018. ⟨119⟩

[114] C. Lin. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*, pages 74–81, 2004. ⟨54⟩

[115] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio. A Structured Self-attentive Sentence Embedding. In *Proc. ICLR*, 2016. ⟨45⟩

[116] B. Liu, X. Lu, and J. S. Culpepper. Strong natural language query generation. *Inf. Retr.*, 2021. ⟨52⟩

[117] B. Liu, H. Zamani, X. Lu, and J. S. Culpepper. Generalizing Discriminative Retrieval Models using Generative Tasks. In *Proc. WWW*, pages 3745–3756, 2021. ⟨52⟩

[118] S. Liu, E. Johns, and A. J. Davison. End-To-End Multi-Task Learning With Attention. In *Proc. CVPR*, pages 1871–1880, 2019. ⟨115⟩

[119] T. Liu. Learning to Rank for Information Retrieval. *Found. Trends in Inf. Ret.*, 3(3):225–331, 2007. ⟨29⟩

[120] T. Liu, J. Huang, W. Zhang, Y. Sun, and H. Wang. Improving Entity Recommendation with Search Log and Multi-Task Learning. In *Proc. IJCAI*, pages 4107–4114, 2018. ⟨115⟩

[121] X. Liu, J. Gao, X. He, L. Deng, K. Duh, and Y. Wang. Representation Learning Using Multi-Task Deep Neural Networks for Semantic Classification and Information Retrieval. In *Proc. NAACL-HLT*, pages 912–921, 2015. ⟨115⟩

[122] I. Loshchilov and F. Hutter. Decoupled Weight Decay Regularization. In *Proc. ICLR*, 2019. ⟨36, 127⟩

[123] X. Lu, O. Kurland, J. S. Culpepper, N. Craswell, and O. Rom. Relevance Modeling with Multiple Query Variations. In *Proc. ICTIR*, pages 27–34, 2019. ⟨5, 74⟩

[124] T. Luong, H. Pham, and C. D. Manning. Effective Approaches to Attention-based Neural Machine Translation. In *Proc. EMNLP*, pages 1412–1421, 2015. ⟨45⟩

[125] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, 2008. ISBN 978-0-521-86571-5. ⟨21, 60⟩

[126] M. McGill. An Evaluation of Factors Affecting Document Ranking by Information Retrieval Systems. Technical report, Syracuse Univ., NY. School of Information Studies, 1979. ⟨14⟩

[127] D. Metzler and W. B. Croft. A Markov Random Field Model for Term Dependencies. In *Proc. SIGIR*, pages 472–479, 2005. ⟨25⟩

[128] D. Metzler and W. B. Croft. Latent concept expansion using markov random fields. In *Proc. SIGIR*, pages 311–318, 2007. ⟨62⟩

[129] D. Metzler, T. Strohman, Y. Zhou, and W. B. Croft. Indri at TREC 2005: Terabyte Track. In *Proc. TREC*, page 7, 2005. ⟨58⟩

[130] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. In *Proc. ICLR*, 2013. ⟨37, 49⟩

[131] B. Mitra and N. Craswell. An Introduction to Neural Information Retrieval. *Found. Trends in Inf. Ret.*, 13(1):1–126, 2018. ⟨27⟩

[132] B. Mitra, F. Diaz, and N. Craswell. Learning to Match using Local and Distributed Representations of Text for Web Search. In *Proc. WWW*, pages 1291–1299, 2017. ⟨52⟩

[133] B. Mitra, S. Hofstatter, H. Zamani, and N. Craswell. Conformer-Kernel with Query Term Independence for Document Retrieval. *arXiv:2007.10434 [cs]*, 2020. ⟨144⟩

[134] A. Moffat and J. Zobel. Rank-biased precision for measurement of retrieval effectiveness. *ACM Trans. Inf. Syst.*, 27(1):2, 2008. ⟨17, 18⟩

[135] A. Moffat, P. Bailey, F. Scholer, and P. Thomas. Assessing the Cognitive Complexity of Information Needs. In *Proc. ADCS*, pages 97–100, 2014. ⟨14⟩

[136] H. R. Mohammad, K. Xu, J. Callan, and J. S. Culpepper. Dynamic Shard Cutoff Prediction for Selective Search. In *Proc. SIGIR*, pages 85–94, 2018. ⟨67⟩

[137] A. Montazeralghaem, H. Zamani, and J. Allan. A Reinforcement Learning Framework for Relevance Feedback. In *Proc. SIGIR*, pages 59–68, 2020. ⟨13⟩

[138] J. Muramatsu and W. Pratt. Transparent Queries: Investigation users' mental models of search engines. In *Proc. SIGIR*, pages 217–224, 2001. ⟨84, 88⟩

[139] R. Nallapati. Discriminative models for information retrieval. In *Proc. SIGIR*, pages 64–71, 2004. ⟨113, 114⟩

[140] R. Nallapati, B. Zhou, C. dos Santos, Ç. Gùlçehre, and B. Xiang. Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond. In *Proc. CoNLL*, pages 280–290, 2016. ⟨87, 95⟩

[141] A. Navon, I. Achituve, H. Maron, G. Chechik, and E. Fetaya. Auxiliary Learning by Implicit Differentiation. In *Proc. ICLR*, page 20, 2021. ⟨115⟩

[142] A. Ng and M. Jordan. On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes. In *Proc. NeurIPS*, 2001. ⟨112⟩

[143] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. In *Proc. NeurIPS (CoCo)*, page 10, 2016. ⟨17, 34, 49, 124⟩

[144] K. Nishida, I. Saito, A. Otsuka, H. Asano, and J. Tomita. Retrieve-and-Read: Multi-task Learning of Information Retrieval and Reading Comprehension. In *Proc. CIKM*, pages 647–656, 2018. ⟨115⟩

[145] R. Nogueira and K. Cho. Passage Re-ranking with BERT. *arXiv:1901.04085 [cs]*, 2019. ⟨27, 50, 52, 110, 113, 120, 125, 127⟩

[146] R. Nogueira and J. Lin. From doc2query to docTTTTTquery, 2019. ⟨86, 87, 93, 94, 118⟩

[147] R. Nogueira, W. Yang, K. Cho, and J. Lin. Multi-Stage Document Ranking with BERT. *arXiv:1910.14424 [cs]*, 2019. ⟨29, 50, 125, 127⟩

[148] R. Nogueira, W. Yang, J. Lin, and K. Cho. Document Expansion by Query Prediction. *arXiv:1904.08375 [cs]*, 2019. ⟨52, 86, 87, 126⟩

[149] R. Nogueira, Z. Jiang, R. Pradeep, and J. Lin. Document Ranking with a Pretrained Sequence-to-Sequence Model. In *Proc. EMNLP (Findings)*, pages 708–718, 2020. ⟨50⟩

[150] P. Ogilvie, J. Callan, and J. Callan. Combining Document Representations for Known-Item Search. In *Proc. SIGIR*, pages 143–150, 2003. ⟨64, 84⟩

[151] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli. Fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In *Proc. NAACL*, pages 48–53, 2019. ⟨89, 90⟩

[152] R. Padaki, Z. Dai, and J. Callan. Rethinking Query Expansion for BERT Reranking. In *Advances in Information Retrieval*, pages 297–304, 2020. ⟨84⟩

[153] H. Padigela, H. Zamani, and W. B. Croft. Investigating the Successes and Failures of BERT for Passage Re-Ranking. *arXiv:1905.01758 [cs]*, 2019. ⟨120⟩

[154] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. *Stanford InfoLab*, 1999. ⟨29⟩

[155] L. Pang, Y. Lan, J. Guo, J. Xu, J. Xu, and X. Cheng. DeepRank: A New Deep Architecture for Relevance Ranking in Information Retrieval. In *Proc. CIKM*, pages 257–266, 2017. ⟨52⟩

[156] K. Papineni, S. Roukos, T. Ward, and W. Zhu. BLEU: A method for automatic evaluation of machine translation. In *Proc. ACL*, pages 311–318, 2002. ⟨54⟩

[157] R. Paulus, C. Xiong, and R. Socher. A Deep Reinforced Model for Abstractive Summarization. In *Proc. ICLR*, 2018. ⟨87, 95⟩

[158] J. Pennington, R. Socher, and C. Manning. Glove: Global Vectors for Word Representation. In *Proc. EMNLP*, pages 1532–1543, 2014. ⟨38⟩

[159] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep Contextualized Word Representations. In *Proc. NAACL-HLT*, pages 2227–2237, 2018. ⟨38⟩

[160] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proc. SIGIR*, pages 275–281, 1998. ⟨23, 25, 110, 113, 118⟩

[161] R. Pradeep, R. Nogueira, and J. Lin. The Expando-Mono-Duo Design Pattern for Text Ranking with Pretrained Sequence-to-Sequence Models. *arXiv:2101.05667 [cs]*, 2021. ⟨16⟩

[162] T. Qin, X. Zhang, M. Tsai, D. Wang, T. Liu, and H. Li. Query-level loss functions for information retrieval. *Inf. Process. Manag.*, 44(2):838–855, 2008. ⟨32⟩

[163] N. Quoc Viet Hung, N. T. Tam, L. N. Tran, and K. Aberer. An Evaluation of Aggregation Techniques in Crowdsourcing. In *Proc. WISE*, pages 1–15, 2013. ⟨98⟩

[164] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020. ⟨50, 87, 93, 94, 115, 123, 126⟩

[165] M. Ranzato, S. Chopra, M. Auli, and W. Zaremba. Sequence Level Training with Recurrent Neural Networks. In *Proc. ICLR*, 2016. ⟨88⟩

[166] S. Robertson and H. Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends in Inf. Ret.*, 3(4):333–389, 2009. ⟨110⟩

[167] S. Robertson, H. Zaragoza, and M. Taylor. Simple BM25 Extension to Multiple Weighted Fields. In *Proc. CIKM*, pages 42–49, 2004. ⟨63⟩

[168] S. E. Robertson. The probabilistic character of relevance. *Inf. Process. Manag.*, 13(4):247–251, 1977. ⟨19, 109⟩

[169] S. E. Robertson. The probability ranking principle in IR. *Journal of documentation*, 33(4): 294–304, 1977. ⟨13, 14⟩

[170] S. E. Robertson and K. S. Jones. Relevance weighting of search terms. *J. Assoc. Inf. Sci. Technol.*, 27(3):129–146, 1976. ⟨113⟩

[171] S. E. Robertson and S. Walker. Microsoft Cambridge at TREC-9: Filtering track. In *Proc. TREC*, 2000. ⟨4, 74⟩

[172] S. E. Robertson, C. J. van Rijsbergen, and M. F. Porter. Probabilistic models of indexing and searching. In *Proc. SIGIR*, pages 35–56, 1980. ⟨113⟩

[173] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, M. Gatford, et al. Okapi at TREC-3. *Nist Special Publication Sp*, 109:109, 1995. ⟨22⟩

[174] J. Rocchio. Relevance Feedback in Information Retrieval. *The Smart retrieval system-experiments in automatic document processing*, pages 313–323, 1971. ⟨4, 60⟩

[175] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv:1609.04747 [cs]*, 2017. ⟨35⟩

[176] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 978-0-262-68053-0. ⟨28⟩

[177] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. ⟨38, 41⟩

[178] B. Salehi, F. Liu, T. Baldwin, and W. Wong. Multitask Learning for Query Segmentation in Job Search. In *Proc. ICTIR*, pages 179–182, 2018. ⟨115⟩

[179] G. Salton. *Automatic Information Organization and Retrieval.* McGraw Hill Text, 1968. ISBN 978-0-07-054485-7. ⟨9⟩

[180] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manag.*, 24(5):513–523, 1988. ⟨109⟩

[181] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975. ⟨21⟩

[182] T. Saracevic and P. Kantor. A study of information seeking and retrieving. III. Searchers, searches, and overlap. *J. Assoc. Inf. Sci. Technol.*, 39(3):197–216, 1988. ⟨14, 15⟩

[183] J. Schmidhuber. Deep learning in neural networks. *Neural Networks*, 61(C):85–117, 2015. ⟨28⟩

[184] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *Proc. ICLR*, 2016. ⟨54⟩

[185] A. See, P. J. Liu, and C. D. Manning. Get To The Point: Summarization with Pointer-Generator Networks. In *Proc. ACL*, pages 1073–1083, 2017. ⟨87, 94, 95⟩

[186] J. A. Shaw and E. A. Fox. Combination of Multiple Searches. In *Proc. TREC*, pages 243–252, 1994. ⟨14, 15⟩

[187] D. Sheldon, M. Shokouhi, M. Szummer, and N. Craswell. LambdaMerge: Merging the results of query reformulations. In *Proc. WSDM*, page 795, 2011. ⟨15, 73, 86⟩

[188] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. Learning semantic representations using convolutional neural networks for web search. In *Proc. WWW*, pages 373–374, 2014. ⟨52⟩

[189] V. B. Sinha, S. Rao, and V. N. Balasubramanian. Fast Dawid-Skene: A Fast Vote Aggregation Scheme for Sentiment Classification. In *KDD WISDOM*, page 8, 2018. ⟨98⟩

[190] F. Song and W. B. Croft. A general language model for information retrieval. In *Proc. CIKM*, pages 316–321, 1999. ⟨25, 73⟩

[191] A. Sordoni, Y. Bengio, H. Vahabi, C. Lioma, J. Grue Simonsen, and J. Nie. A Hierarchical Recurrent Encoder-Decoder for Generative Context-Aware Query Suggestion. In *Proc. CIKM*, pages 553–562, 2015. ⟨86⟩

[192] M. Srikanth and R. Srihari. Biterm language models for document retrieval. In *Proc. SIGIR*, pages 425–426, 2002. ⟨25⟩

[193] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic Attribution for Deep Networks. In *Proc. ICML*, pages 3319–3328, 2017. ⟨139, 142⟩

[194] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to Sequence Learning with Neural Networks. In *Proc. NeurIPS*, pages 3104–3112, 2014. ⟨40, 118⟩

[195] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 2018. ISBN 978-0-262-03924-6. ⟨92⟩

[196] J. Teevan. The re:search engine: Simultaneous support for finding and re-finding. In *Proc. UIST*, pages 23–32, 2007. ⟨86⟩

[197] P. Thomas, B. Billerbeck, N. Craswell, and R. W. White. Investigating Searchers' Mental Models to Inform Search Explanations. *ACM Trans. Inf. Syst.*, 38(1):10:1–10:25, 2019. ⟨84, 88⟩

[198] K. Toutanova, C. Brockett, K. M. Tran, and S. Amershi. A Dataset and Evaluation Metrics for Abstractive Compression of Sentences and Short Paragraphs. In *Proc. EMNLP*, pages 340–350, 2016. ⟨96⟩

[199] H. Turtle and W. B. Croft. Evaluation of an inference network-based retrieval model. *ACM Trans. Inf. Syst.*, 9(3):187–222, 1991. ⟨14⟩

[200] V. N. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998. ISBN 978-0-471-03003-4. ⟨114⟩

[201] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proc. NeurIPS*, pages 6000–6010, 2017. ⟨27, 36, 41, 43, 45, 46, 90, 110⟩

[202] C. C. Vogt and G. W. Cottrell. Fusion Via a Linear Combination of Scores. *Information Retrieval*, 1(3):151–173, 1999. ⟨15⟩

[203] E. Voorhees, T. Alam, S. Bedrick, D. Demner-Fushman, W. R. Hersh, K. Lo, K. Roberts, I. Soboroff, and L. L. Wang. TREC-COVID: Constructing a pandemic information retrieval test collection. *ACM SIGIR Forum*, 54(1):1:1–1:12, 2021. ⟨49⟩

[204] E. M. Voorhees. Overview of the TREC 2004 Robust Retrieval Track. In *Proc. TREC*, page 10, 2004. ⟨17⟩

[205] E. M. Voorhees. Overview of the TREC 2005 Robust Retrieval Track. In *Proc. TREC*, 2005. ⟨57⟩

[206] E. Walker and A. S. Nowacki. Understanding Equivalence and Noninferiority Testing. *J. Gen. Intern. Med.*, 26(2):192–196, 2011. ⟨73⟩

[207] J. Wang, L. Yu, W. Zhang, Y. Gong, Y. Xu, B. Wang, P. Zhang, and D. Zhang. IRGAN: A Minimax Game for Unifying Generative and Discriminative Information Retrieval Models. In *Proc. SIGIR*, pages 515–524, 2017. ⟨114⟩

[208] W. Webber, A. Moffat, and J. Zobel. A similarity measure for indefinite rankings. *ACM Trans. Inf. Syst.*, 28(4):20:1–20:38, 2010. ⟨73, 79⟩

[209] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3):229–256, 1992. ⟨54, 92⟩

[210] R. J. Williams and D. Zipser. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Comput.*, 1(2):270–280, 1989. ⟨41, 91⟩

[211] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proc. EMNLP (Demos)*, pages 38–45, 2020. ⟨51⟩

[212] B. Wu, C. Xiong, M. Sun, and Z. Liu. Query Suggestion with Feedback Memory Network. In *Proc. WWW*, pages 1563–1571, 2018. ⟨86⟩

[213] F. Xia, T. Liu, J. Wang, W. Zhang, and H. Li. Listwise Approach to Learning to Rank - Theory and Algorithm. In *Proc. ICML*, pages 1192–1199, 2008. ⟨32⟩

[214] C. Xiong and J. Callan. Query expansion with Freebase. In *Proc. ICTIR*, pages 111–120, 2015. ⟨62, 63⟩

[215] C. Xiong, Z. Dai, J. Callan, Z. Liu, and R. Power. End-to-End Neural Ad-hoc Ranking with Kernel Pooling. In *Proc. SIGIR*, pages 55–64, 2017. ⟨52⟩

[216] J. Xu and W. B. Croft. Quary expansion using local and global document analysis. In *Proc. SIGIR*, pages 4–11, 1996. ⟨60⟩

[217] Y. Xu, G. J. Jones, and B. Wang. Query Dependent Pseudo-relevance Feedback Based on Wikipedia. In *Proc. SIGIR*, pages 59–66, 2009. ⟨62, 63⟩

[218] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, and S. Mannor. Learn What Not to Learn: Action Elimination with Deep Reinforcement Learning. In *Proc. NeurIPS*, pages 3562–3573, 2018. ⟨90⟩

[219] H. Zamani, M. Dehghani, W. B. Croft, E. Learned-Miller, and J. Kamps. From Neural Re-Ranking to Neural Ranking: Learning a Sparse Representation for Inverted Indexing. In *Proc. CIKM*, pages 497–506, 2018. ⟨52⟩

[220] H. Zamani, B. Mitra, X. Song, N. Craswell, and S. Tiwary. Neural Ranking Models with Multiple Document Fields. In *Proc. WSDM*, 2018. ⟨64⟩

[221] H. Zaragoza, N. Craswell, M. Taylor, S. Saria, and S. Robertson. Microsoft Cambridge at TREC–13: Web and HARD tracks. In *Proc. TREC*, page 7, 2004. ⟨22, 58, 63, 67⟩

[222] Y. Zeng and J. Nie. A Simple and Efficient Multi-Task Learning Approach for Conditioned Dialogue Generation. In *Proc. NAACL-HLT*, pages 4927–4939, 2021. ⟨120⟩

[223] C. Zhai. Statistical Language Models for Information Retrieval A Critical Review. *Found. Trends in Inf. Ret.*, 2(3):137–213, 2008. ⟨48⟩

[224] C. Zhai and J. Lafferty. A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval. In *Proc. SIGIR*, pages 334–342, 2001. ⟨24⟩

[225] Y. Zhang and Q. Yang. A Survey on Multi-Task Learning. *arXiv:1707.08114 [cs]*, 2018. ⟨115⟩

[226] Z. Zheng, K. Hui, B. He, X. Han, L. Sun, and A. Yates. BERT-QE: Contextualized Query Expansion for Document Re-ranking. In *Proc. EMNLP (Findings)*, pages 4718–4728, 2020. ⟨13⟩

[227] N. Zhiltsov, A. Kotov, and F. Nikolaev. Fielded Sequential Dependence Model for Ad-Hoc Entity Retrieval in the Web of Data. In *Proc. SIGIR*, pages 253–262, 2015. ⟨64⟩

[228] S. Zou, G. Tao, J. Wang, W. Zhang, and D. Zhang. On the Equilibrium of Query Reformulation and Document Retrieval. In *Proc. ICTIR*, pages 43–50, 2018. ⟨114⟩