Learning to Rank in the Age of Muppets

by

Chengcheng Hu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

I hereby declare that I am the sole author of works related to RQ2 in this thesis and are not written for any publications. The research works related to RQ1 are conducted by Yue Zhang, Chengcheng Hu and Yuqi Liu under the supervision of Hui Fang and Jimmy Lin. The RQ1 related works are published in sustaiNLP 2021. As co-authors of works related to RQ1, I am responsible for contributing coding the learning to rank module and writing the published paper.

Citations:

Yue Zhang, ChengCheng Hu, Yuqi Liu, Hui Fang, and Jimmy Lin. 2021. Learning to Rank in the Age of Muppets: Effectiveness–Efficiency Tradeoffs in Multi-Stage Ranking. In Proceedings of the Second Workshop on Simple and Efficient Natural Language Processing, pages 64–73, Virtual. Association for Computational Linguistics.

Abstract

The emergence of BERT in 2018 has brought a huge boon to retrieval effectiveness in many tasks across various domains and led the recent research landscape of IR to transformer-related technologies. While researchers are fascinated by the power of BERT, along with related transformer models, substantial computational costs incurred by transformers become an unavoidable problem. Meanwhile, under the light of BERT, there are "out-of-date" but fairly effective techniques forgotten by people. For example, learning to rank was one of the most popular technologies a decade ago.

In this work, we aim to answer two research questions: RQ1 is whether using learning to rank as a filtering stage in a multi-stage reranking pipeline can improve the efficiency of reranking using transformers without sacrificing effectiveness. In addition, we are interested in if using transformer-based features in the traditional learning to rank framework can increase effectiveness as RQ2.

To answer RQ1, we implement a multi-stage reranking pipeline which places learning to rank as a filter in the middle stage. This configuration allows the pipeline to only send the most promising candidates using cheap learning to rank module to expensive neural rerankers, hence a speedup in overall latency for transformer-based reranking can be obtained without a degradation in effectiveness. By applying the pipeline on MS MARCO passage and document ranking tasks, we can achieve up to 18 $\times$ increase in efficiency while maintaining the same level of effectiveness. Moreover, our method is orthogonal to other techniques that focus on neural models themselves to accelerate inference. Hence, our method can be combined with other accelerating works to further save computational costs and latency.

For RQ2, since transformers generate relevance scores for different query-document pairs independently, it is possible to use transformer-based scores as learning to rank features, so that learning to rank can take advantage of transformers to increase retrieval effectiveness. Applied to the MS MARCO passage and document ranking tasks, we gain a maximal 52% increase in effectiveness by adding the BERT-based feature compared to the "traditional" learning to rank. Also, we obtain a result with a little bit higher effectiveness by adding transformer-based features with other traditional features in learning to rank, compared to the standard retrieve-and-rerank design with transformers.

This work explores potential roles of learning to rank in the age of muppets.[1] In a broader sense, this work illustrates that we should stand on the shoulder of giants, which is what we learned and discovered in history, to explore next unknowns.

[1] Muppets being a whimsical way to refer to BERT and related transformer models.

Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Jimmy Lin. Although we have never met each other in person due to the pandemic, I did not feel unfamiliar with him at all as he was always kind and supportive for the past 2 years. I can still clearly recall the first time I ask to be his undergraduate research assistant. The reproduction task on MS MARCO passage retrieval led me into the world of IR. At that time, I knew nothing about IR and NLP; amazingly, 2 years after that, I am going to graduate with this thesis. I would want to thank him again for mentoring me in the research area with all his knowledge and experience and inspiring me with his enthusiasm over the past 2 years.

I would also like to thank my readers, Professor Gordon Cormack and Professor Jian Zhao, for their time reviewing my thesis and providing precious comments.

I would like to thank Yue Zhang, a PhD candidates in University of Delaware, and Yuqi Liu, a master student of Jimmy Lin, for all the time we spent together for learning to rank. Also it is a pity that because of Covid, I was not able to see dsg group members in lab, it is my pleasure to know all talent researchers in dsg group.

Last but not least, I want to thank my parents for their continuous care and love for all years.

Dedication

This is dedicated to my parents for their unconditional love and caring.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Pretrained transformers have achieved a huge success in improving retrieval effectiveness in various tasks across different domains, e.g., text ranking, machine translation. Among various transformers, BERT [11], which stands for Bidirectional Encoder Representations from Transformers, is one of the best-known examples. Results provided by BERT are undoubtedly superior to results generated by traditional ranking methods used before. Many established and replicable experiments make this superiority widely accepted. Outside of the academic research field, BERT also brings benefits to industry. Google and Microsoft have made posts about applying BERT to their products improves search experience to customers [23].

While BERT has significantly increased retrieval effectiveness, large computational costs and query latencies are incurred by the standard "retrieve-and-rerank" setup with transformers. The growing costs are obstacles to using neural rerankers in real-world applications [52]. For example, the ColBERT paper [19] shows that reranking 1000 hits from the MS MARCO passage dataset takes 32.9 seconds per query using the BERT_{large} model. Also, researchers realize the high computational costs of transformer-based rerankers [15] compel the exploration of other approaches, for example, simplified models [16, 42, 30, 28, 14, 17] and learned dense representations [50].

Naturally, reducing computational costs and latencies while maintaining the same level of effectiveness would be a practical problem for academic research and industry to further take advantage of BERT and other transformers. Instead of focusing on the cutting-edge BERT reranker itself, if doing some retrospectives, we wonder if some "out-of-date" methods may offer help to BERT.

Back in the 2010s, before the rise of transformers, learning to rank was one of the major

developments in text ranking. Learning to rank emerged at the intersection of machine learning, information retrieval, and natural language processing. Documents have many signals to show whether it is relevant or not to a given query. Each signal can be represented as a feature. Learning to rank utilizes hand-crafted, manually engineered features to automatically build ranking models with supervised machine learning techniques [21].

The processing time of learning to rank is relatively short compared to transformers. Although BERT is so powerful that it covers up lights of all other methods, the retrieval effectiveness obtained by learning to rank is not poor.

Now, the problem becomes if there exist any forms of cooperation between BERT and learning to rank to help us be as effective as BERT and as efficient as possible. In a broader sense, we should stand on the shoulders of giants, history, to explore next unknowns.

1.1 Research Question

As there are two parties involved, BERT and learning to rank, the cooperation could naturally go in both ways. To be specific, the two ways of cooperation are (1) How can learning to rank help BERT? (2) How can BERT help learning to rank? From here, two research questions are motivated, respectively.

1.1.1 RQ1

Treating BERT as the main part, we wonder what can learning to rank offer to help BERT reduce computational costs and query latencies while maintaining BERT effectiveness. There are two factors affecting the total runtime of BERT in a retrieve-and-rerank design: the number of query-document candidate pairs and the processing time for each individual pair.

BERT requires a much longer time than other techniques when reranking the same number of candidates, because its sophisticated neural network architecture costs longer per query-document pair in terms of latencies. The ColBERT paper [19] listed that the number of FLOPs/query $BERT_{large}$ need is as 570,000 $\times$ more than KNRM [48, 49]. There are methods used to directly accelerate inference, e.g., knowledge distillation, early exits, model simplification. However, we wish not to change the nature of BERT in this work, i.e., per query latencies will not change.

The option left is to reduce the number of candidates which are sent to BERT for inferencing. Obviously, without any extra actions, only sending fewer hits to BERT will

harm effectiveness. However, inspired by the multi-stage reranking architecture, we can first narrow down the range of true relevant documents and send more promising candidates under the consideration of BERT to maintain the effectiveness. Here comes the usage of cheap learning to rank reranker and leads the Research Question 1 (RQ1), can we use learning to rank as a filtering stage in a multi-stage reranking pipeline to improve the efficiency of BERT reranking without sacrificing effectiveness?

1.1.2 RQ2

Considering the opposite way of cooperation in RQ1, we are curious about what advantages BERT can bring to learning to rank in terms of effectiveness.

Features in learning to rank are scores, calculated by different statistics or language models, which represent similarities between queries and documents. Meanwhile, BERT ranks candidates based on scores for each pair of query and document, which are generated independently of the context [11]. It is reasonable to come up with the idea of considering transformer-based features, including but not limited to BERT and TCT-ColBERT scores, etc., as features within the learning to rank framework, where gradient boosted tree [29] is one of the common choices of frameworks. In this way, we can utilize relevance signals from BERT, which have higher effectiveness, to favour inference in learning to rank framework.

Inspired by the above idea, we address Research Question 2 (RQ2) here: can we use BERT-based features in traditional learning to rank framework to increase effectiveness?

1.2 Contributions

The contribution of this thesis is summarized below.

- Answer "YES" to RQ1.

  By inserting the cheaper learning to rank technique as a "filtering" stage in a multi-stage ranking architecture, fewer candidates need to be sent for further BERT inferences, hence reducing the processing time for BERT without degradation in accuracy. With experiments we conducted on MS MARCO passage and document, we find we can maintain the same level of effectiveness as a standard retrieve-and-rerank method using BERT while we can obtain an up to 18 $\times$ speedup in terms of latency. Furthermore, this design can help us control the effectiveness-efficiency trade-offs and make applications more practical in the real world [52].

- Answer "YES" to RQ2.

  We have verified that learning to rank with transformer-based features can largely increase the original effectiveness of learning to rank by up to 52%. Furthermore, using transformer-based and other traditional features has a little bit better accuracy compared to the standard retrieve-and-rerank with BERT configuration. Learning to rank with transformer-based and other traditional features does not yield a huge improvement on effectiveness compared to directly using transformers. When transformer-related relevance signals are present, contributions from other features become minimal. Specifically, for passage, the effectiveness of learning to rank with the BERT-based feature is $\sim$1 point higher than the standard transformer effectiveness and meanwhile, for document, there is a 0.2 point win in effectiveness using our learning to rank module with the BERT-based feature.

1.3 Thesis Organization

The thesis is organized as follows:

- Chapter 2 reviews fundamental background and existing works of learning to rank, BERT in detail.
- Chapter 3 introduces multi-stage pipelines used for comparisons in experiments and the learning to rank architecture built with Pyserini for this work, including used features and the preprocessing steps on the queries and documents.
- Chapter 4 introduces datasets and the evaluation metrics, then describes the experimental setup and implementations.
- Chapter 5 presents experimental results and analysis.
- Chapter 6 concludes this thesis and discusses potential directions for future works.

Chapter 2

Background and Related Work

2.1 Learning to Rank

Learning to rank was one of the most popular techniques in the Information Retrieval and Natural Language Processing community before the era of muppets. Back to that period of time, learning to rank had outstanding retrieval effectiveness as it utilizes machine learning techniques. As learning to rank falls into the supervised learning category, it involves 2 steps: training and testing. Figure 2.1 visualizes the whole process.

In the training process, the training set includes query and document pairs. The relevance between the query and document is given by a label, usually a number. The higher the number is, the more relevant the document is to the given query. Then, the feature extraction phase happens. During feature extraction, a feature vector $x_{i,j}$, which is computed from a feature function $\phi(q_i, d_{i,j})$, is created for each query-document pair. The learning algorithm takes feature vectors and labels as input to train a ranking model $f(x)$ [21].

When it comes to the testing process, unseen queries are used in the test set. Feature vectors are created for each unseen query-document pair again. Then, the trained ranking model $f(x)$ will assign scores to each query-document pair and give the final rank list based on sorted scores.

The ranking model can be generalized since its learning is based on features. In other words, the learning ability is positively related to the qualities of chosen features. In previous works, LETOR [38] dataset uses 46 features, which includes a few commonly used ranking features, for example, query-independent features such as document length; query-dependent features, which involves interactions between query and document, like TFIDF, BM25 [39], etc..

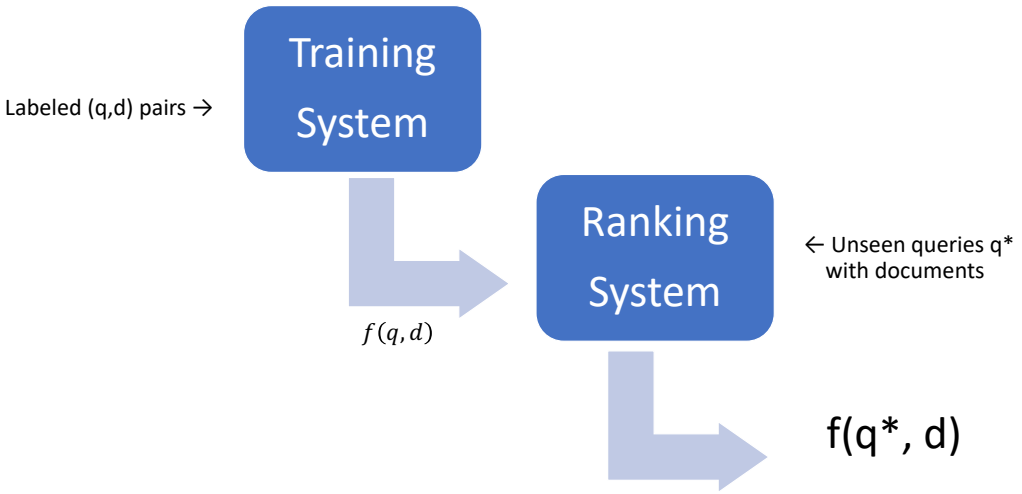

Figure 2.1: Learning to Rank Architecture

Evaluation metrics are calculated by comparing the ranked list output with relevance judgements. Widely used metrics are adopted, such as MRR (Mean Reciprocal Rank), MAP (Mean Average Precision), NDCG (Normalized Discounted Cumulative Gain).

There are three main approaches mentioned in existing learning to rank works: pointwise approach, pairwise approach, and listwise approach. The pointwise approach treats the ranking problem as a classification/regression task. Each query-document pair is learned and predicted by the ranking model independently in the training and testing process. Meanwhile, the pairwise approach focuses on the ranking order of two query-document pairs. Ranking SVM [18], RankBoost, and lambdaMART [7] are common pairwise approaches. The listwise approach naturally takes ranking lists as input in both the training and testing process and outputs the permutation of inputs based on optimization strategy in the chosen algorithm. ListNet is an instance of listwise approaches. These three approaches mainly differ in their input/output structures and loss functions.

More examples for each approach can be seen in Table 2.1. Normally, the pairwise and listwise approaches outperform the pointwise approach. In the most recent Yahoo Learning to Rank Challenge [9], the team from Microsoft using LambdaMART [8], which belongs to the pairwise approach, is the top winner.

Categories	Examples				
Pointwise	OC SVM	McRank	Prank		
Pairwise	Ranking SVM	RankBoost	RankNet	LambdaMART	LambdaRank
Listwise	SVM MAP	AdaRank	ListNet	SoftRank	AppRank

Table 2.1: Examples for each Learning to Rank Methods Categories

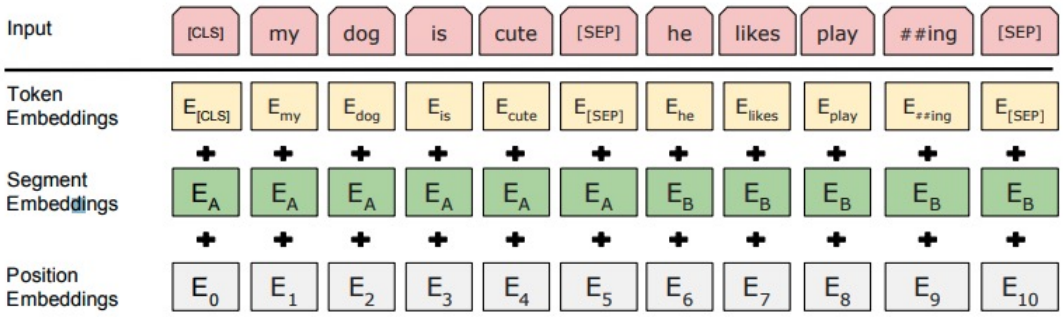

Figure 2.2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings. The Diagram is taken from Devlin et al. [10]

.

The major difficulty of using learning to rank is that learning to rank relies heavily on hand-crafted feature engineering.

2.2 BERT

BERT stands for Bidirectional Encoder Representations from Transformers [10], which is a pretrained transformer-based deep learning technique. Once BERT was published in 2018, it immediately amazed researchers by its performance and became the state of the art. BERT takes a sequence of tokens as input and then outputs a sequence of contextual embeddings. Since BERT provides context-dependent representations of the input tokens, more sophisticated characteristics of language, such as semantics and syntax, can be learned by the model. This makes BERT be able to interpret meanings in a complicated but more realistic linguistic context.

Specifically, BERT adopts a uniform standard for input format in order to make BERT available for various downstream tasks. The input format has two special characters. The first special token is [CLS]: every input sequence to BERT starts with [CLS]. Another special delimiter token [SEP] is appended to the end of each input sentence, so that BERT can handle a sequence with more than one sentence. BERT uses WordPiece embeddings [46] with a 30,000 token vocabulary to tokenize original texts into subwords. In the final input representation, each token contains the subword token embedding, the segment embeddings (indicates which sentence contains the given token) and the position embeddings. There is a length limitation of 512 tokens for BERT. The way of input construction is also illustrated in Figure 2.2.

As a pretrained transformer, BERT involves two phases: pre-training and fine-tuning. There are two different tasks in pre-training of BERT. The first is Masked Language Model (MLM). Given a sentence, 15% of input tokens are randomly masked. MLM is to train a deep bidirectional representation that can predict those masked tokens from both left and right contexts. The second task is Next Sentence Prediction (NSP), which aims to find the relationship between sentences. The input is a pair of sentences A and B. There is a 50% of chance that B is the next sentence following A; while another 50% of chance is that B is a randomly selected sentence from the corpus. NSP trains a model to predict whether B is the correct following sentence of A. Fine-tuning for downstream tasks is relatively inexpensive compared to pre-training. Pretrained models (also called model checkpoints) can be downloaded. Task-specific inputs and outputs are then plugged into those models to fine-tune all parameters from end-to-end.

In Jan 2019, BERT was applied to text ranking problems for the first time [34] on the MS MARCO passage ranking tasks. With BERT, the effectiveness obtained a ∼30% relative gain, compared to the best pre-BERT method.

2.3 Effectiveness-Efficiency Trade-offs

Effectiveness of the system usually refers to the quality of the output. Common effectiveness metrics in IR are MRR@10, NDCG@20, and MAP etc.. On the other hand, efficiency focuses on completing a task using fewer resources. In IR, retrieval latencies and computational costs are usually considered in efficiency problems. Effective technologies usually come with higher computational costs. In order to achieve higher quality results, more and more sophisticated ranking architectures are adopted, resulting in a slower system. Especially when the world enters the neural era, the problem of simultaneously increasing effectiveness and efficiency in end-to-end retrieval systems becomes trickier.

Wang [44] proposed using multi-stage retrieval architectures to control the trade-offs between effectiveness and efficiency. Specifically, multi-stage retrieval architecture can lower the latency by optimizing recalls in previous stages and sending only the most promising candidates to the final expensive techniques. In fact, there are already many researches using multi-stage neural pipelines in the context of transformers [34, 42, 28, 36].

There are various ways to further reduce computational costs. The first approach is to transform big and exhaustive models into smaller or simpler models to reduce inference time. We can transfer knowledge learned from a larger model into a smaller model using distillation. The work of Gao [14] shows that knowledge distillation can successfully help us save a large amount of processing time while only losing minimal effectiveness. There are also other works [30, 27] taking the same approach. Another major approach is to ask the model to exit earlier at an appropriate time to optimize the efficiency [42, 47]. By adding a classifier to each layer of transformers, once a classifier in a middle layer reaches the confidence threshold, the early exit is performed and speedups are obtained.

Chapter 3

Retrieval Architecture

3.1 Multi-Stage Ranking

We adopt a standard formulation of multi-stage ranking. It involves the first stage retrieval H_0, which uses keyword search against an inverted index and then retrieves top k_0 candidates from the corpus. There is a pipeline of rerankers, denoted H_1 to H_N, following the H_0 stage. Each subsequent stage H_i will receive a ranked list with top k_{i-1} hits for each query obtained from the previous stage H_{i-1}, rerank these hits, and then pass the result to the next stage. The ranked list received from the last stage is the final result we used to evaluate by standard tools.

3.1.1 Two designs used to answer RQ1

We compare two designs of multi-stage ranking architectures to answer RQ1.

BoW + BERT As a baseline, we adopt the retrieve-and-rerank approach originally proposed by Nogueira and Cho [31], which is widely used as the standard architecture for applying pretrained transformers to ranking. k_0 is used to denote the number of candidates from bag-of-words retrieval. We notate a configuration of this design as BoW(k_0) + BERT, with a commonly used default $k = 1000$ [31].

Furthermore, we introduce the docTTTTTquery document expansion technique (d2q variant from now on) [35, 33] based on predicting queries for which a text would be relevant. We denote the BoW retrieval on documents with their original texts concatenated by the predicted queries as BoW_{d2q} for d2q variants and the pipeline as BoW_{d2q} + BERT.

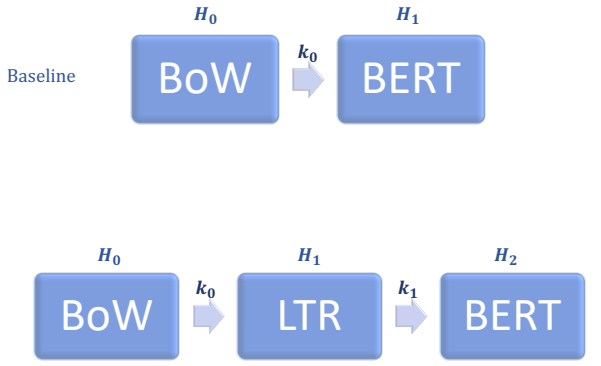

Figure 3.1: Multi-stage architectures

BoW + LTR + BERT This represents our proposed design of inserting a filtering stage before BERT to reduce the number of candidates for further neural inferencing. We introduce k_0, same as before, the number of candidates from bag-of-words retrieval and k_1, the number of top hits of the new ranking generated by our learning to rank stage. We notate a configuration of this design as BoW(k_0) + LTR(k_1) + BERT. k_1 top hits from the learning to rank stage are sent to the final BERT stage.

We utilize the advantage of d2q in our learning to rank stage as well. We use LTR_{d2q} to represent the setting in which the learning to rank stage extracts features from the document expansions as an extra field.

Given this setup, the answer to RQ1 becomes an effectiveness-efficiency trade-off of BoW + BERT vs. BoW+LTR+BERT (d2q variant), depicted in Figure 3.1. We want to know, when the same level of effectiveness is required, how much latency can we save in the final stage BERT by passing fewer candidates to consider. To obtain an answer, we use the learning to rank module to extract features from documents and train a model using supervised learning, where the goal is to maximize recall. Then by applying the trained model to the ranked list returned by bag-of-words retrieval, we can pass more valuable candidates to the expensive BERT reranking stage. Note that we use BERT in our notation for convenience. BERT can be replaced with other neural transformers, such as T5.

3.1.2 Design used to answer RQ2

To answer RQ2, we compare the following designs of multi-stage ranking architectures.

BoW + LTR This retrieve-and-rank approach involves BoW and the learning to rank technique. Similarly, we use k_0 to denote the number of candidates returned by bag-of-words retrieval and sent to the learning to rank stage. To make a fair comparison with the common default setup, we choose k_0 to be 1000 as well. Different from the learning to rank stage above with lots of traditional non-neural features, we include neural features which use scores generated by BERT and other transformers this time. Here, we introduce new notations of different sets of features in the learning to rank module.

- **LTR** refers to a set of features that contains all features in term-based, score-based, proximity-based and translation-based scores as described in Section 3.2.
- **LTR w/ BERT**$_{only}$ represents the learning to rank module that only has one feature, which is the BERT score feature.
- **LTR w/ BERT** means the model has features both in **LTR** and **LTR w/ BERT**$_{only}$, i.e., all traditional features plus a neural transformer feature.

By comparing BoW+LTR and BoW + LTR w/ BERT, we can answer RQ2 by a potential effectiveness improvement in BoW + LTR w/ BERT configuration. We want to know whether given the same list of candidates, can learning to rank achieve a higher level of effectiveness with the help from BERT. In addition, we compare BoW+BERT vs. BoW + LTR w/ BERT to explore whether learning to rank with transformer-based and all traditional features can outperform the standard retrieve-and-rerank pipeline with transformers.

3.2 Learning-to-Rank Features

We split learning to rank features we implemented into five categories: term-based, score-based, proximity-based, translation-based, and transformer-based [52]. These features are inspired by previous studies on learning to rank [37, 12]. The summary of features is in Table 3.1. An enumeration of all features is listed in Appendix A.

Feature Category	#	Examples
Term-based	54	Max of IDF, Max of TF
Score-based	14	BM25, DFR
Proximity-based	15	Co-occurrences, BM25-TP
Translation-based	4	translation probability
Transformer-based	2	BERT

Table 3.1: Summary of learning to rank features.

3.2.1 Term-based Features

Based on previous works on learning to rank, we compute various term statistics, such as term frequency (TF), inverse document frequency (IDF), log term probability, and inverse collection term frequency. We also use relevance scores of each term, computed by existing retrieval modes such as BM25, as term-based features combining different types of statistics. We use six aggregation functions to aggregate term-based statistics for all terms in a query: max, min, sum, mean, median, and the ratio between max and min.

3.2.2 Score-based Features

In order to further improve retrieval accuracy, we have taken document-level statistics such as document length into consideration besides term-based statistics. We also include traditional bag-of-words retrieval models (e.g., BM25, Query Likelihood, Divergence From Randomness) as features, since they are effective when combining term-based and document-based statistics. Note that computing the retrieval model score is equivalent to using sum as the aggregation function for term-based statistics.

3.2.3 Proximity-based Features

While proximity among terms captures an important relevance signal, traditional retrieval models cannot catch the signal since they assume terms are independent and ignore their relationships. Hence, we include features such as the counts of ordered and unordered co-occurrence of bigrams within different window sizes, which directly capture the proximity of query terms, i.e. the distance between terms within documents. We also include proximity-based retrieval functions, such as SDM [29] and BM25-TP, as our features.

3.2.4 Translation-based Features

So far, all the features we used focus on exact match between query terms and document terms. However, semantic relationships between a query and a document usually play a crucial role in improving retrieval effectiveness. We use a translation model IBM Model 1 [6, 5] to measure the translation probability between queries and documents, in order to incorporate features that capture semantic relationships between a query and a document and resolve the vocabulary gap. With the IBM Model 1 translation model, we compute the conditional translation probability $P(q|D)$ for a query token and a document pair, and then take the product of all individual conditional query token probabilities as our final query-document feature. To be specific, we adopt the following equation in our IBM Model1 feature.

$$P(Q|D) = \prod_{q \in Q} P(q|D) \tag{3.1}$$

$$P(q|D) = (1 - \lambda) \left[\sum_{d \in D} T(q|d) P(d|D) \right] + \lambda \cdot P(q|C) \tag{3.2}$$

3.2.5 Transformer-based Features

When using transformers as standard rerankers, BERT for example, we compute the probability that the passage is relevant to a query by sending the $[CLS]$ vector as input to a single layer neural network. The relevance probability for a query-passage pair is calculated independently [31]. We use these relevance probabilities as one of our feature scores, so that the learning to rank algorithm utilizes the power of transformers. Note that this type of features is only used to answer RQ2.

All learning to rank features are extracted at the level of tokens. Specifically, both queries and tokens are tokenized into a multi-field representation: (1) *raw* the field contains the original tokens; (2) *stemmed* the field consists of stemmed tokens; (3) *subword* the field breaks tokens into subwords; (4) *d2q* the field includes stemmed tokens of docTTTTTquery predictions (only used when d2q variants are involved). Feature extraction is performed over all applicable fields for each query-document pair. Each field has 83 different features. The *raw* and *subword* fields have extra four translation-based features. Transformer-based features only work on one field since they use data from external resources and will have no difference on each field. Table 3.1 summarizes features with examples for each feature category.

Chapter 4

Experimental Setup

4.1 Data

The MS MARCO passage dataset [3] is used for training and testing. The training set contains approximately ∼500K queries. The development and test sets contain ∼7K queries each. Detail statistics are listed in Table 4.1. On average, each query has one relevant passage; negative passages are taken from BM25 results that are not otherwise judged as relevant. In order to make the training process more efficient, there is no need to use all negative passages. The final training data contains queries with all positive passages and 20 negative passages for each query.

The MS MARCO document dataset [3] is also used for evaluation in a zero-shot manner. Each document is segmented into multiple passages as the neural models cannot process long documents. The sliding window strategy [36] is adopted in segmentation step. Specifically, the window length is 3 sentences with a stride of 1 sentence, so that the segments length is close to the passage length. Retrieval is performed at the segment level, as well as the learning to rank stage and transformers. The final document score is the highest relevance score among its segments, which is known as the MaxP method [4, 10, 1, 41].

4.2 Implementation

Anserini [51], an open-source IR toolkit built on Lucene, is used to build the indexes and perform the first stage retrieval. To retrieve top-ranked candidate passages in the first

	Passage		Document
	train	dev	dev
# of documents	8.8M		3.2M
# of queries	502,939	6,980	5,193

Table 4.1: MS MARCO dataset statistics.

Paramete	Value
num_leaves	200
learning_rate	0.1
min_data_in_leaf	50
max_bin	255
max_depth	-1
min_sum_hessian_in_leaf	0
feature_fraction	1
learning_rate	0.1
num_boost_round	1000
early_stopping_round	200

Table 4.2: LambdaMART Algorithm Parameters.

stage, BM25 is used. The choice of parameters ($k_1 = 0.82$ and $b = 0.68$) is made to optimize recall@1000 based on the authors' recommendations.[1] The retrieved candidate passages are then sent to the feature extraction phase through Pyserini, which is the Python interface of Anserini [22].

The LambdaMART algorithm, implemented in the LightGBM library, is adopted in the learning to rank module. LambdaMART is a modified LambdaNet but replaces the underlying neural network model with gradient boosted regression trees. Hyperparameters are tuned to optimize recall@200 on the MS MARCO dev set using grid search. Specifically, num_leaves is 200, learning_rate is 0.1, min_data_in_leaf is 50, max_bin is 255. We fix early stopping patience to 200 and use up to 1000 trees. The full list of parameters is listed in Table 4.2. In order to make learning to rank inference more efficient, we adopt batch processing and multi-threading to help us leverage multi-core CPUs.

[1] https://github.com/castorini/anserini/blob/master/docs/experiments-msmarco-passage.md

We run the final stage transformer rerankers in the open source project PyGaggle, which provides a gaggle of deep neural architectures for text ranking and question answering.[2] We use checkpoints provided by PyGaggle for BERT-large and T5-base models, fine-tuned on the MS MARCO passage data, as this work does not explore the final stage neural transformers. The checkpoints for those two models have been shown to have competitive effectiveness as baselines [34, 32, 36]. In order to align with the transformers' requirements, all token sequences in the batch are set to 512 tokens. If their lengths exceed 512 tokens, we truncate the part after the 512th token.

4.3 Latency Measurement

To answer RQ1, we are interested in the trade-offs between effectiveness and efficiency. In terms of efficiency, we focus on the per query latencies here. We have two different servers to run the multi-stage architecture. The first server is used to run the first-stage retrieval and the learning to rank filtering stage. This server is equipped with 2 Intel Xeon Platinum 8160 CPUs and the index is stored on a local SSD partition. We perform transformer inference on another 6 core server with a single Tesla V100 GPU. To note that we exclude the processing time to load data and models and sum component results to compute end-to-end query latency. To make the comparison stand out, we also normalize latencies from different pipelines into a speedup value.

[2] https://github.com/castorini/pygaggle

Chapter 5

Results and Analysis

In this chapter, we present experimental results. Those experiments are designed to answer RQ1 and RQ2. By results we obtained from rigorous experiments, we can provide positive answers to both research questions.

5.1 Results for RQ1

In this section, we are going to compare the pipelines mentioned in Section 3.1.1, which is BoW + BERT vs. BoW + LTR + BERT (d2q variant when applicable), in the main result tables. Furthermore, we have done a few analytical experiments on the trade-off relationship and feature importance.

5.1.1 RQ1 results for passage ranking

In Table 5.1, evaluation results on MS MARCO passage ranking is presented. There are various kinds of configurations based on the notation introduced in Section 3.1.1. We report MRR@10 and NDCG@10 as our effectiveness and also latency (s/query) to show efficiency for each type of configuration. We highlight the number N in the Table 5.1, where N is the number of candidates sent to the most expensive neural reranking stage. As the neural reranking stage consumes most of the total time, smaller N leads to a shorter total time. We want to emphasize that in Table 5.1, our goal is to obtain effectiveness parity between our proposed configuration with the baselines while having a faster inference process.

	Configuration	N	MRR@10	NDCG@10	Latency	
(a)	BoW(1k) + BERT	1000	0.379	0.441	9.63s	
(b)	BoW(10k) + LTR(100) + BERT	100	0.381	0.443	1.32s	(7×)
(c)	BoW(10k) + LTR$_{d2q}$(20)+ BERT	20	0.382	0.442	0.53s	(18×)
(d)	BoW(1k) + T5	1000	0.380	0.443	5.60s	
(e)	BoW(10k) + LTR(100) + T5	100	0.382	0.445	0.92s	(6×)
(f)	BoW(10k) + LTR$_{d2q}$(20) + T5	20	0.382	0.444	0.46s	(12×)
(g)	BoW$_{d2q}$(1k) + BERT	1000	0.389	0.454	9.63s	
(h)	BoW$_{d2q}$(10k) + LTR$_{d2q}$(50) + BERT	50	0.389	0.454	0.83s	(12×)
(i)	BoW$_{d2q}$(1k) + T5	1000	0.386	0.453	5.60s	
(j)	BoW$_{d2q}$(10k) + LTR$_{d2q}$(50) + T5	50	0.388	0.454	0.63s	(9×)

Table 5.1: The effectiveness and efficiency of different pipeline configurations on the MS MARCO passage ranking task. The effectiveness of the pipelines with additional LTR modules is statistically indistinguishable from the baselines without the LTR modules.

Rows (a), (d), (g), (i) are baseline configurations, with a common N=1000. We have six learning to rank pipelines presented. We choose the smallest N, i.e., the smallest number of candidates that need to be sent to transformers, that can achieve the same level of effectiveness as the baselines. The k_0, which is the number of candidates from the first-stage retrieval, is different. Although in our proposed configuration k_0 is larger than the baseline, since they only use traditional retrieval techniques, the increase in the first and second stage ranking time is negligible, which can be explained in a detailed latency composition table later. We conduct two-tailed paired t-tests to verify that no significant effectiveness differences exist between results before and after inserting learning to rank as the filtering stage. With the help of learning to rank filtering, we only need to use N from 20 to 100 for the expensive transformer inference, based on different configuration setups, to achieve the same or higher MRR@10 and NDCG@10. We also report per query latency for each configuration, along with a speedup value which is normalized against the latency of baselines. From the last column, we observe that we can achieve 18 × speedup with the help of learning to rank's d2q variant version. We obtain 10 × speedup on average for different configurations, and vary from 6 × to 18 × depending on the neural model we pick and whether we use the d2q variant. Note that the BoW + BERT baseline used here in row (a) is a more efficient version, which has adopted batch tokenization and other engineering optimizations, compared to the version reported by Khattab and Zaharia [19].

	Model	Retrieval	Feature Extraction	LTR Prediction	Neural Reranking	**Total**
(a)	BoW(1k) + BERT	15	-	-	9610	**9630**
(b)	BoW(10k) + LTR(100) + BERT	120	180	40	980	**1320**
(c)	BoW(1k) + T5	15	-	-	5580	**5600**
(d)	BoW(10k) + LTR(100) + T5	120	180	40	584	**920**

Table 5.2: Detailed breakdown of latency (ms/query) for a few representative pipeline configurations on the MS MARCO passage ranking task.

Hence, the speedup values presented in Table 5.1 is compared to a well-optimized baseline.

In order to show a detailed latency composition, Table 5.2 is presented for 4 configurations. Note that, we only use learning to rank without the d2q variant as an example since d2q is applied to the corpus as a preprocessing step, which will not affect query latency at search time. Obviously, we observe that the final stage of neural ranking takes dominant effect in the total latency and the neural ranking run time is linearly dependent on N, which is the number of candidates sent to neural transformers in Table 5.1. This confirms our method to minimize N achieves better efficiency. Although increasing k_0 in the first stage and also feature extraction and prediction steps in the learning to rank filtering stage introduces overheads, this is worthwhile. By comparing rows (a) and (b) in Table 5.2, increasing 300 ms/query overhead by introducing learning to rank and larger k_0 can help us save 9000 ms/query (30 $\times$ overhead) in the final stage. The total overhead remains modest. The T5 model originally takes less time than BERT, hence the overhead of learning to rank is relatively larger than BERT and the speedup is not as large as BERT.

Feature extraction is the most expensive step in learning to rank. However, the time in feature extraction is not much related to the number of features. In fact, experiments show that the number of features in the learning to rank module will not make a big difference in the time of the feature extraction phase. This is because the most time-consuming part of feature extraction is loading the forward index for each document. In the design of the learning to rank module, no matter how many features we want, we only need to load the document once; once the document is in cache, each single feature extraction is very fast. For now, the implementation of feature extraction is relatively inefficient in Java code. One potential future direction for this work would be expending more engineering effort, for example, optimization proposed by Asadi and Lin [2], to further decrease overhead introduced by learning to rank.

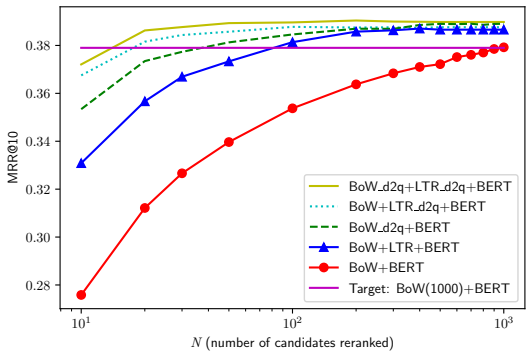

Figure 5.1: MRR@10 as a function of N, the number of candidates reranked by the final neural reranker, on the MS MARCO passage ranking task.

Figure 5.1 displays MRR@10 as a function of the number N on the x-axis (log scale) for our five configurations compared to BERT. Similar to Table 5.1, N represents the number of hits sent to the most expensive neural transformers. Since the latency of neural inference is dominant, speedups are mostly obtained from smaller N, i.e., the smaller N normally means less total latency. The target, which is the MRR@10 of baseline BoW(1000) + BERT, is drawn horizontally in purple. From the figure, we can see configurations with learning to rank as the middle filtering stage helps to achieve the target MRR@10 with smaller N. Moreover, with the help of d2q expansions, N can be further reduced without degradation in MRR@10. Note that in order to make learning to rank perform better, k_1, the number of hits from first stage retrieval, is increased. However, it will not affect much in terms of efficiency, as the time in first stage retrieval is minimal compared to BERT inference.

Figure 5.2 is another way to represent the relationship between the MRR@10 and N, as we used latencies for different N on the x axis. Note that we only draw a subset of Figure 5.1 where the x axis only has latencies when $N \leq 300$. This figure shows a suitable latency range to choose learning to rank. We can see it is not worthwhile to use learning to rank when the desired latency is less than 0.5s, since the blue curve is not much above the red curve. It is because learning to rank itself introduces overhead. Table 5.2 shows that learning to rank overhead is 0.34s for a query with 10k hits. When the desired latency is only around 0.5s, there is little time left for neural inference, hence we are not able to take many advantages from neural transformers. It is worthwhile to spend all the time on BERT directly when the desired latency is small, as BERT is much more powerful in terms of effectiveness compared to learning to rank. For reference, when spending 0.34s on

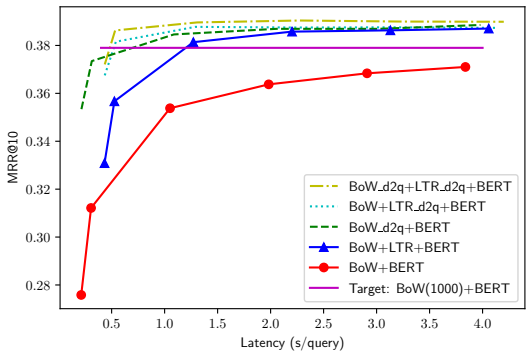

Figure 5.2: MRR@10 vs. reranking latency on the MS MARCO passage ranking task.

BoW+LTR, the MRR@10 is 0.28; on the other hand, we can reach MRR@10 0.28 when running BoW+BERT in 0.43s.

So far, all the tables and figures focus on the condition when we reach effectiveness parity. Figure 5.3 uses BoW+LTR+BERT configuration as an example to show possible trade-offs between effectiveness and efficiency. Note that speedups are normalized by using BoW(1000) + BERT as the reference point. There will be times that a faster processing speed is desired while the highest accuracy is not necessary. From figure 5.3, we can see various possible speedups with their corresponding MRR@10. For example, we can achieve 0.36 MRR@10 using only $N = 20$ as our setting and enjoy 17 $\times$ speedup without using d2q (see Table 5.1).

As mentioned in Chapter 3, there are four categories of features used in the experiment to answer RQ2. We are interested in how much contribution each feature category makes to the final effectiveness, hence we conduct ablation experiments to investigate the importance of each category and list results in Table 5.3. From the table, we can see that the score-based feature category is the least important one. A possible explanation is that score-based features have a lot of redundant information which is also revealed in term-based features. Score-based features and term-based features are only performing different aggregation functions on the same raw signals, such as term frequency, document frequency, etc. As score-based features include only summation on signals and term-based including max, min, mean, median and the ratio between max and min, term-based features have a larger impact on the learning to rank.

Obviously, translation-based features have the dominant effect in the model. Without translation-based features, the effectiveness drops the most while the translation-based

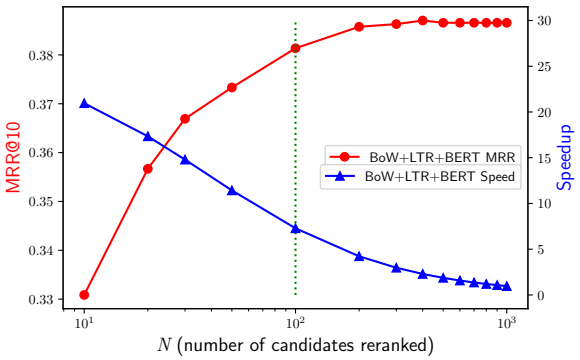

Figure 5.3: MRR@10 vs. speedup for different N, the number of candidates reranked by the final neural reranker, on the MS MARCO passage ranking task.

category has the least features. All other three feature categories are computed based on either exact match between query and document or static signals in query and document. Translation-based features support semantic matching by modelling alternative expressions of query terms so that a document can match a query term that does not appear in the document, i.e., solve the problem known as the vocabulary gap.

Furthermore, we have used built-in functions in the LightGBM library to analyze the importance of each feature. Specifically, the function ranks importance by the total number of splits and the total gain of splits. Aligning with the result from the ablation experiments, all translation-based features are placed in the top 10 features with the highest number of splits. This reconfirms that it is important for our module that translation-based features bridge the gap between queries and documents by semantic matching. Also, traditional bag-of-words retrieval models still form the base of learning to rank, since the feature with the highest total gain of splits is the GL2 model from the Divergence From Randomness family of scoring functions.

5.1.2 RQ1 results for document ranking

To examine the generality and robustness of our proposed configuration, we also applied our learning to rank pipeline on the MS MARCO document ranking task. Note that all experiments are conducted in a zero-shot manner. We split documents into segments that are 3 sentences long with a stride of 1 sentence. All the middle steps, from first stage retrieval, learning to rank filtering to neural ranking, are performed on segments. The final

Configuration	Recall@100	MRR@10
Full Model	0.781	0.382
− Score-based	0.780 (−0.09%)	0.381 (−0.24%)
− Term-based	0.771 (−1.27%)	0.379 (−0.74%)
− Proximity-based	0.767 (−1.76%)	0.379 (−0.96%)
− Translation-based	0.743 (−4.82%)	0.371 (−3.01%)

Table 5.3: Feature importance ablation results on the MS MARCO passage ranking task.

	Configurlation	N	MRR@100	NDCG@10	Latency	
(a)	BoW(1k) + BERT	1000	0.3658	0.4252	9.35s	
(b)	BoW(10k) + LTR(100) + BERT	100	0.3661	0.4288	1.32s	(7 ×)

Table 5.4: The effectiveness and efficiency of different pipeline configurations on the MS MARCO document ranking task. The effectiveness of the pipelines with additional LTR modules is statistically indistinguishable from the baselines without the LTR modules.

stage transformers and learning to rank model are trained on MS MARCO passage only. We use MaxP [4, 10, 1, 41] at the very last step to evaluate effectiveness. We still achieve the same level of desired speedups as experiments on passages without lower MRR@10.

Results are shown in Table 5.4, which is organized in the same structure as Table 5.1. Recall again that we aim to achieve effectiveness parity with respect to the baseline (BoW+BERT in this case). Due to the configuration of d2q, this time we are unable to append suitable d2q predictions to segments, hence the speedup is not as fast as results in the passage experiment. However, for configuration without the d2q variant, BoW+LTR+BERT can obtain the same level of speedups as in passage ranking. This confirms that our learning to rank module is extendable to other datasets.

5.2 Results for RQ2

In this section, we are going to compare pipelines mentioned in Section 3.1.2, which is BoW + LTR vs. BoW + LTR w/ BERT and BoW + BERT vs. BoW + LTR w/BERT, on the MS MARCO passage and document ranking tasks in the main result tables. Furthermore, we have analyzed feature importance on new proposed learning to rank models.

		MRR@10	
		Baselines	
(a)	BoW + LTR	0.248	
(b)	BoW + BERT	0.365	
(c)	BoW + TCT-ColBERT	0.352	
		Our configurations	
(d)	BoW + LTR$_{w/BERT_{only}}$	0.373	
(e)	BoW + LTR$_{w/BERT}$	0.378	(+ 52%)*
(f)	BoW + LTR$_{w/TCT-ColBERT_{only}}$	0.348	
(g)	BoW + LTR$_{w/TCT-ColBERT}$	0.355	(+ 43%)*

Table 5.5: The effectiveness of different pipeline configurations on the MS MARCO passage. *Relative increase percentages are calculated with respect to row (a).

5.2.1 RQ2 results for passage ranking

Table 5.5 shows results for different configurations on MS MARCO passage. Row (a) serves as the baseline which is having learning to rank with only traditional features as a reranker on top of bag-of-words retrieval. Note that our goal is to have our proposed learning to rank model with neural features obtaining higher effectiveness, compared to the baseline. We also list rows (b) and (c) as references for results from standard transformer reranking experiments, specifically results for passages are found in the TCT-ColBERT paper [24]. Rows (b) and (c) serve as baselines of standard retrieve-and-rerank with transformers. Rows (d) - (g) are our proposed configurations with different sets of features. Specifically, row (d) is the experiment result of learning to rank with only the BERT-based feature in the feature set; while row (e) contains all other traditional features. Rows (f) and (g) are organized in a similar manner as (d) and (e), except that they interact with TCT-ColBERT scores.

We report MRR@10 for passage and calculate the percentage increase in effectiveness for rows (e) and (f) with respect to corresponding traditional learning to rank baselines. We use row (a) as our baseline to calculate the increase percentage in effectiveness for both row (e) and (g). We observe that the BERT-based feature helps the learning to rank module increase its effectiveness by 52% in passage ranking. A 43% increase in MRR@10 for incorporating TCT-ColBERT can be seen. The results from these two rows confirm that transformer-based features can bring significant benefits to the effectiveness of learning

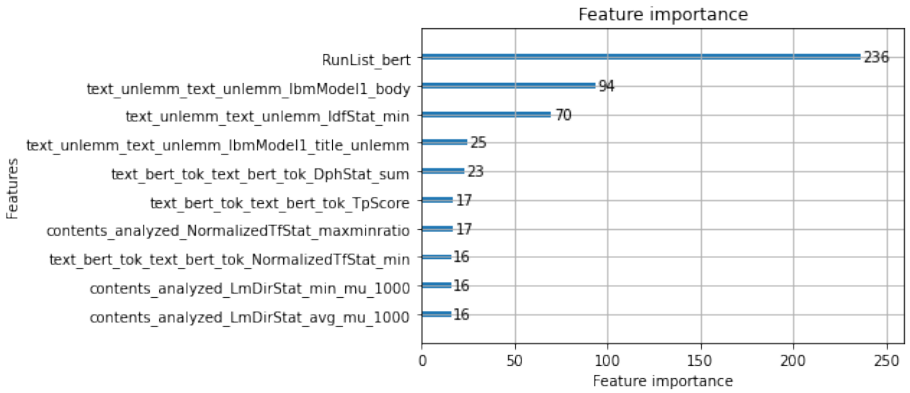

Figure 5.4: Feature importance analysis for learning to rank with BERT scores based on split

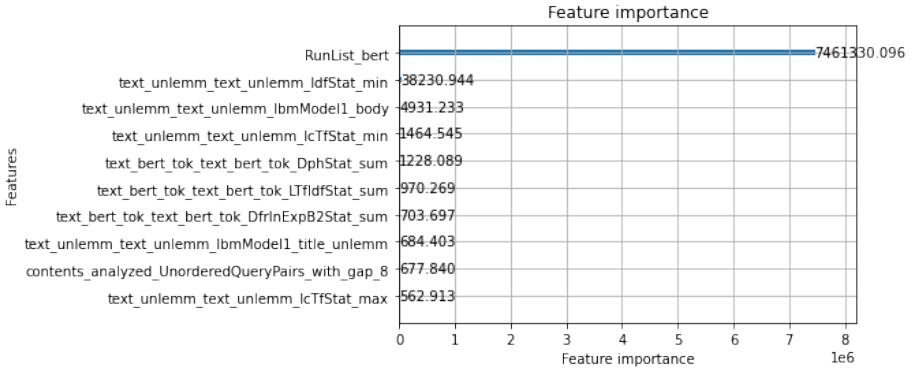

Figure 5.5: Feature importance analysis for Learning to rank with BERT scores based on gain

		MRR@100	
		Baselines	
(a)	BoW + LTR	0.309	
(b)	BoW + BERT	0.370	
		Our configurations	
(d)	BoW + $\text{LTR}_{w/BERT_{only}}$	0.366	
(e)	BoW + $\text{LTR}_{w/BERT}$	0.372	(+ 20%)*

Table 5.6: The effectiveness of different pipeline configurations on the MS MARCO document. *Relative increase percentage is calculated with respect to row (a).

to rank. Furthermore, by comparing rows (b) and (e), there is little increase by adding traditional features to the learning to rank model with transformer-based features, which illustrates that the contribution from traditional features is minimal when the transformer-based features are present.

We show the result of feature importance analysis using built-in functions in LightGBM in Figures 5.4 and 5.5. Only the top 10 features with the most impact are presented in figures. In Figure 5.4, LightGBM ranks the importance by the number of splits for that feature, which is the x axis. In other words, more splits mean that the feature is used more frequently in the model, hence the feature is more important. In Figure 5.5, the total gains of splits that use the feature determine the importance. Both figures agree that the BERT-based feature takes the dominant position in the learning to rank model.

5.2.2 RQ2 results for document ranking

Table 5.6 shows results for experiments on MS MARCO document and is organized in a same manner as Table 5.5. We want to emphasize again that we adopt sliding window and MaxP strategy to split document corpus into segments, i.e., learning to rank model and the final neural models are all trained on MS MARCO passage only; also learning to rank inferencing and neural inferencing are performed on the segment level.

We can see similar results as in passage. Adding traditional features on learning to rank model with neural features helps a little in effectiveness; on the other hand, adding BERT feature on top of traditional features increases 20% in terms of retrieval effectiveness. Though the increase is not as competitive as results for passage, it is still significant. A

possible explanation is that since we did not train a learning to rank model on MS MARCO document, the strength of learning to rank model is weaker, hence less impressive increase in effectiveness.

Chapter 6

Conclusion and Future Work

When BERT was published in 2018 for the first time, it immediately impressed researchers in the IR and NLP fields with its superior retrieval ability. By then, BERT became the state of the art and led the major research direction in recent years. Many research works have demonstrated that the "retrieve-and-rerank" approach with transformers largely increased the retrieval effectiveness compared to traditional, non-neural ranking methods.

Meanwhile, expensive costs come along with neural approaches as transformers are intrinsically complicated. Growing costs impede the usage of BERT in real-world applications. Also, the light of BERT is so bright that it covers all the old but fairly effective traditional ranking approaches. Some thrown-away methods are still valuable and worth further exploration. Based on these concerns, this thesis raises two research questions. The RQ1 is that is it possible to use learning to rank, which was the most popular technique before the arrival of BERT, as a filtering stage, so that efficiency can be increased by sending only the most promising candidates to expensive BERT reranker. While the RQ2 is interested in if BERT-based features can still benefit learning to rank approach in terms of effectiveness.

To answer RQ1, this thesis performs experiments with BoW+LTR+BERT pipelines on MS MARCO passage and document. It turns out that with this BoW+LTR+BERT configuration, we can achieve an up to 18 $\times$ speedup while maintaining the same level of accuracy. Also, by using the BoW+LTR+BERT configuration, we can control the effectiveness-efficiency trade-offs by choosing the number of candidates which are sent to BERT.

The answer for RQ2 is also "Yes". By adding transformer-based features in the learning to rank technique, we observe a significant increase in effectiveness, which is up to 52%.

Moreover, using learning to rank with transformer-based features brings us a little increase in effectiveness compared to using transformers alone.

These answers demonstrate that there are still potential roles for the learning to rank technique, even if it is "out-of-fashion", in the age of muppets dominated by transformers and other neural models. Our field should not open a new page without remembering our history.

There are a few potential directions for future works. The first is to extend experiments to more available datasets. Also, other researchers have proposed acceleration approaches directly dealing with neural models. As this thesis proposes to resolve the efficiency problem in an orthogonal way to the approach in RQ1, we can try to combine our method with those approaches to accumulate speedups.

References

[1] Zeynep Akkalyoncu Yilmaz, Wei Yang, Haotian Zhang, and Jimmy Lin. Cross-domain modeling of sentence-level evidence for document retrieval. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3490–3496, Hong Kong, China, 2019.

[2] Nima Asadi and Jimmy Lin. Document vector representations for feature extraction in multi-stage document ranking. *Information Retrieval*, 16(6):747–768, 2013.

[3] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. *arXiv:1611.09268v3*, 2018.

[4] Michael Bendersky and Oren Kurland. Utilizing passage-based language models for ad hoc document retrieval. *Information Retrieval*, 13:157–187, 2009.

[5] Leonid Boytsov and Zico Kolter. Exploring classic and neural lexical translation models for information retrieval: Interpretability, effectiveness, and efficiency benefits. *arXiv:2102.06815*, 2021.

[6] Leonid Boytsov and Eric Nyberg. Flexible retrieval with NMSLIB and FlexNeuART. *arXiv:2010.14848*, 2020.

[7] Christopher Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11, 01 2010.

[8] Christopher Burges, Krysta Svore, Paul Bennett, Andrzej Pastusiak, and Qiang Wu. Learning to rank using an ensemble of lambda-gradient models. *Journal of Machine Learning Research - Proceedings Track*, 14:25–35, 01 2011.

[9] Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In Olivier Chapelle, Yi Chang, and Tie-Yan Liu, editors, *Proceedings of the Learning to Rank Challenge*, volume 14 of *Proceedings of Machine Learning Research*, pages 1–24, Haifa, Israel, 25 Jun 2011. PMLR.

[10] Zhuyun Dai and Jamie Callan. Deeper text understanding for IR with contextual neural language modeling. In *Proceedings of the 42nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2019)*, pages 985–988, Paris, France, 2019.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019.

[12] Luke Gallagher, Antonio Mallia, J. Shane Culpepper, Torsten Suel, and B. Barla Cambazoglu. Feature extraction for large-scale text collections. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, pages 3015–3022, 2020.

[13] Yasser Ganjisaffar, Rich Caruana, and Cristina Videira Lopes. Bagging gradient-boosted trees for high precision, low variance ranking models. In *Proceedings of the 34rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2011)*, pages 85–94, Beijing, China, 2011.

[14] Luyu Gao, Zhuyun Dai, and Jamie Callan. Understanding BERT rankers under distillation. In *Proceedings of the 2020 ACM SIGIR International Conference on Theory of Information Retrieval (ICTIR 2020)*, 2020.

[15] Sebastian Hofstätter and Allan Hanbury. Let's measure run time! Extending the IR replicability infrastructure to include performance aspects. In *Proceedings of the Open-Source IR Replicability Challenge (OSIRRC 2019): CEUR Workshop Proceedings Vol-2409*, pages 12–16, Paris, France, 2019.

[16] Sebastian Hofstätter, Markus Zlabinger, and Allan Hanbury. Interpretable & time-budget-constrained contextualization for re-ranking. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, Santiago de Compostela, Spain, 2020.

[17] Jyun-Yu Jiang, Chenyan Xiong, Chia-Jung Lee, and Wei Wang. Long document ranking with query-directed sparse transformer. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4594–4605, 2020.

[18] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.

[19] Omar Khattab and Matei Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2020)*, pages 39–48, 2020.

[20] Bernhard Kratzwald and Stefan Feuerriegel. Adaptive document retrieval for deep question answering. 2018.

[21] Hang Li. *Learning to Rank for Information Retrieval and Natural Language Processing*. Morgan Claypool Publishers, San Rafael, CA, 2011.

[22] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2021)*, pages 2356–2362, 2021.

[23] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. Pretrained transformers for text ranking: BERT and beyond. *arXiv:2010.06467*, 2020.

[24] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. Distilling dense representations for ranking using tightly-coupled teachers. *ArXiv*, abs/2010.11386, 2020.

[25] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. Distilling dense representations for ranking using tightly-coupled teachers. *arXiv:2010.11386*, 2020.

[26] Xiaolu Lu, Alistair Moffat, and J. Shane Culpepper. On the cost of extracting proximity features for term-dependency models. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 293–302, 2015.

[27] Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. Efficient document re-ranking for transformers by pre-computing term representations. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2020)*, pages 49–58, 2020.

[28] Yoshitomo Matsubara, Thuy Vu, and Alessandro Moschitti. Reranking for efficient transformer-based answer selection. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2020)*, pages 1577–1580, 2020.

[29] Donald Metzler and W. Bruce Croft. A Markov random field model for term dependencies. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2005)*, pages 472–479, Salvador, Brazil, 2005.

[30] Bhaskar Mitra, Sebastian Hofstätter, Hamed Zamani, and Nick Craswell. Conformer-kernel with query term independence for document retrieval. *arXiv:2007.10434*, 2020.

[31] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with BERT. *arXiv:1901.04085*, 2019.

[32] Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. Document ranking with a pretrained sequence-to-sequence model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 708–718, November 2020.

[33] Rodrigo Nogueira and Jimmy Lin. From doc2query to docTTTTTquery, 2019.

[34] Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. Multi-stage document ranking with BERT. *arXiv:1910.14424*, 2019.

[35] Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. Document expansion by query prediction. *arXiv:1904.08375*, 2019.

[36] Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. The expando-mono-duo design pattern for text ranking with pretrained sequence-to-sequence models. *arXiv:2101.05667*, 2021.

[37] Tao Qin and Tie-Yan Liu. Introducing LETOR 4.0 datasets. *arXiv:1306.2597*, 2013.

[38] Tao Qin and Tie-Yan Liu. Letor: A benchmark collection for research on learning to rank for information retrieval. *ArXiv:1306.2597*, 2013.

[39] S.E. Robertson and S.Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, 1994.

[40] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. In *Proceedings of the 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing at NeurIPS 2019*, Vancouver, British Columbia, Canada, 2020.

[41] Eilon Sheetrit, Anna Shtok, and Oren Kurland. A passage-based approach to learning to rank documents. *Information Retrieval Journal*, 23:159–186, 2020.

[42] Luca Soldaini and Alessandro Moschitti. The cascade transformer: An application for efficient answer sentence selection. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5697–5708, 2020.

[43] N. Tonellotto, C. MacDonald, and I. Ounis. Efficient and effective retrieval using selective pruning. In *Proceedings of the sixth ACM international conference on Web search and data mining*, 2013.

[44] Lidan Wang, Jimmy Lin, and Donald Metzler. A cascade ranking model for efficient ranked retrieval. In *Proceedings of the 34th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2011)*, pages 105–114, Beijing, China, 2011.

[45] Xiao Wang, Yaxiong Wu, Xi Wang, Craig Macdonald, and Iadh Ounis. University of Glasgow Terrier team at the TREC 2020 deep learning track. In *Proceedings of the Twenty-Ninth Text REtrieval Conference (TREC 2020)*, 2020.

[46] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

[47] Ji Xin, Rodrigo Nogueira, Yaoliang Yu, and Jimmy Lin. Early exiting BERT for efficient document ranking. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 83–88, 2020.

[48] Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. Convolutional neural networks for so-matching n-grams in ad-hoc search zhuyun dai. 2017.

[49] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 55–64, New York, NY, USA, 2017. Association for Computing Machinery.

[50] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *arXiv:2007.00808*, 2020.

[51] Peilin Yang, Hui Fang, and Jimmy Lin. Anserini: Reproducible ranking baselines using Lucene. *Journal of Data and Information Quality*, 10(4):1–20, 2018.

[52] Yue Zhang, Chengcheng Hu, Yuqi Liu, Hui Fang, and Jimmy Lin. Learning to rank in the age of muppets: Effectiveness–efficiency tradeoffs in multi-stage ranking. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, 2021.

APPENDICES

Appendix A

Full List of LTR Features

Term-based				
				Sum of ICTF
Average of BM25	Average of DFR_GL2	Average of DFR_in_expB2	Average of DPH	Average of ICTF
Median of BM25	Median of DFR_GL2	Median of DFR_in_expB2	Median of DPH	Median of ICTF
Max of BM25	Max of DFR_GL2	Max of DFR_in_expB2	Max of DPH	Max of ICTF
Min of BM25	Min of DFR_GL2	Min of DFR_in_expB2	Min of DPH	Min of ICTF
MaxMinRatio of BM25	MaxMinRatio of DFR_GL2	MaxMinRatio of DFR_in_expB2	MaxMinRatio of DPH	MaxMinRatio of ICTF
Sum of IDF		Sum of Normalized TF	Sum of TF	
Average of IDF	Average of LMDir	Average of Normalized TF	Average of TF	Average of TFIDF
Median of IDF	Median of LMDir	Median of Normalized TF	Median of TF	Median of TFIDF
Max of IDF	Max of LMDir	Max of Normalized TF	Max of TF	Max of TFIDF
Min of IDF	Min of LMDir	Min of Normalized TF	Min of TF	Min of TFIDF
MaxMinRatio of IDF	MaxMinRatio of LMDir	MaxMinRatio of Normalized TF	MaxMinRatio of TF	MaxMinRatio of TFIDF
Score-based				
SCS	Probablity Sum	Doc Size	Query Length	Query Coverage Ratio
Unique Term Count in Query	Matching Term Count	Normalized TFIDF	BM25	LMDir
DFR_GL2	DFR_in_expB2	DPH	TFIDF	
Proximity-based				
UnorderedSequentialPairs with gap 3	OrderedSequentialPairs with gap 3	UnorderedQueryPairs with gap 3	OrderedQueryPairs with gap 3	BM25-TP
UnorderedSequentialPairs with gap 8	OrderedSequentialPairs with gap 8	UnorderedQueryPairs with gap 8	OrderedQueryPairs with gap 8	Proximity
UnorderedSequentialPairs with gap 15	OrderedSequentialPairs with gap 15	UnorderedQueryPairs with gap 15	OrderedQueryPairs with gap 15	TP distance
Translation-based				
title IBM Model1(*raw* field)	url IBM Model1(*raw* field)	body IBM Model1(*raw* field)	body IBM Model (*subword* field)	
Muppet-based				
BERT	TCT_ColBERT			