Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Магістерська робота

освітній ступінь – магістр

на тему: «**Семантичний пошук на основі представлень, отриманих за допомогою нейронних мереж**»

Виконав: студент 2-го року навчання,
освітньо-наукової програми
«Прикладна математика»,
спеціальності 113 Прикладна математика
Крошин Олександр Андрійович
Керівник: Швай Н.О.
кандидат фіз.-мат. наук, доцент
Рецензент ________________________
(прізвище та ініціали)

Кваліфікаційна робота захищена
з оцінкою ____________________

Секретар ЕК ____________

«____» ____________2022 р.

Київ – 2022

Table of contents

Анотація

Магістерська робота присвячена задачі семантичного пошуку, зокрема, задачі ранжування документів за запитами користувачів. В роботі розглядається розв’язок задачі ранжування за допомогою нейронних мереж архітектури GPT-3. В роботі пропонується розв’язувати цю задачу за допомогою імплементації GPT-3 під назвою OPT.

Розглядаються різні варіанти розв’язку задачі без додаткового тренування, зокрема, використання різних форматів вхідних текстів. В роботі пропонується модель Cross-Encoder на основі OPT, що демонструє кращі результати роботи на прийнятих в індустрії тестах в порівнянні з існуючими рішеннями, що вважаються стандартами. Розглядаються експерименти з дистиляцією моделей OPT задля подальшого покращення їхніх результатів в задачах семантичного пошуку.

Робота складається з трьох розділів. Перший, теоретичний розділ присвячений постановці задачі ранжування, огляду існуючих метрик та класичних підходів. Другий розділ описує використання нейронних мереж в задачі ранжування а також демонструє рішення, побудовані на базі моделей архітектури GPT-3. Третій, практичний розділ присвячений побудові та тренуванню Cross-Encoder на основі моделей OPT, експериментам з дистиляцією моделей та обробці результатів. Результати експериментів демонструються у висновку.

Annotation

This master's thesis is dedicated to semantic search problem. It is focused on the task of re-ranking documents based on user query, particularly, by solving it via applying neural networks based on GPT-3 architecture. This work introduces GPT-3 implementation, OPT, to the re-ranking task, proposes best prompt for its zero-shot evaluation, introduces OPT-based cross-encoder that outperforms previous BERT-based SOTA approaches by a margin and experiments with different knowledge-distillation settings in order to additionally boost performance of smaller model.

The work is split into three sections. The first section sets theoretical background of re-ranking problem, reviews metrics and industry-standard approaches. Second section explains usage of neural networks in semantic search and describes different solutions to re-ranking problem using networks of GPT-3 architecture. The third, experimental section, covers training an OPT-based Cross-Encoder, describes knowledge-distillation experiments and evaluates proposed solutions. Results of the experiments are discussed in conclusion section. All the used literature is in the References section.

Introduction

Recent developments in Natural Language Processing have greatly impacted its neighboring domains such as Information Retrieval. Appearance of large pre-trained language models[1] has changed industry standards and approaches. Text search is among the areas that were highly impacted. This work focuses on text search problem known as re-ranking, where a list of documents has to be sorted by relevance given a specific user query.

Introduction of BERT[2] led to appearance of high-quality vector representations of search queries and document collections that significantly boosted search performance and let to evolution of search systems. Instead of relying on classic solution based on BM-25 algorithm[3], newer search pipelines use high-quality input representations that only a pre-trained language model can create. However, BM-25 didn't become obsolete, as indexing enormous collections remains a significant challenge.

BERT influenced the appearance of two different main neural-based approaches to re-ranking task that can be classified as bi-encoder and cross-encoder[4]. Bi-Encoder approach lies in creating separate vector representations (embeddings) for documents and queries and ranking documents using cosine similarity. Cross-Encoder is a classifier built on top of BERT that uses document-query pair as a single input string and outputs their grade of relevance.

Introduction of GPT-3[5] allowed solving NLP problems out of the box without additional training, and re-ranking task is among these tasks. Muennighoff[6] demonstrated GPT-3 great out-of-the box performance in re-ranking, comparable to SOTA BERT-based solutions[7]. Adopting SOTA approaches to GPT architecture may lead to superior models for semantic search tasks, which is the main motivation of this paper.

The aim of this paper is to utilize recently released GPT-3 implementation OPT[8] for re-ranking problem, create best zero-shot pipeline experimenting with various input prompts, propose OPT-based cross-encoder that outperforms previous state-of-the-art solutions on benchmarks such as BEIR[9] and explore possibilities of creating a distillated version of OPT that would outperform SOTA in both speed and precision.

First part of the paper is focused on an overview of re-ranking problem itself, description of industry-standard solutions and introduction of necessary benchmarks and metrics. Second part provides an overview of GPT network architecture, describes how GPT can be used in a problem of re-ranking, introduces a cross-encoder structure and a variety of methods and techniques that can be used for re-ranking problem. The final, third part, describes training datasets, discusses training of a GPT-based cross-encoder, applies various knowledge distillation techniques required for expanding OPT zero-shot capabilities and analyzes achieved results.

1. Semantic search. Document Re-ranking problem, an overview

1.1. Query-Document matching, Semantic Search, Retrieval and Re-rank.

The problem of matching similar pieces of information based on their relevancy scourges the world of information technologies for decades. Producing relevant search results is a crucial for any information-retrieval system, with search itself being one of the greatest achievements of recent decades.

One could broadly define search as the task of matching a piece of information that the user wants to know with information that the user feeds into the search system as a clue. This problem is also known as retrieval, and given a system that operates a large collection of documents, it may be defined as follows:

*Given a set of documents **D** and a user query **Q**, a system must extract a subset of **N** documents, semantically most relevant to the query.*

A system that operates documents and queries may be called a retrieval system.

1.1.2 Retrieval and Re-ranking

Retrieval systems have certain constraints. List of results must be formed fast, so that the user gets results in real-time, and retrieval on a very large collections of documents leads to inability to use most advanced methods of search in real-time – indexing all the documents is too costly and time-consuming. Industry overcame this problem by introducing a two-stage pipeline called retrieve and re-rank. The first stage, retrieval, extracts a subset of N most relevant documents. Second stage, re-ranking, can be defined as a problem of sorting a subset of N most relevant documents obtained during retrieval stage, in order to provide the user with more accurate search result.

1.1.3 Semantic Search

Classic methods of retrieval and re-ranking are based on statistical matching of word occurrences between documents and queries, whereas more recent approaches are based on an idea of semantically matching document and query using vector representations. A re-ranking pipeline that uses methods based on semantic matching can be called Semantic Search (or Neural Search). It is based on the idea that any piece of text can be represented as an N-dimensional real vector, and a simple function, such as cosine similarity can be used to determine a grade of semantic similarity between two given pieces of text (or other types of information).

1.1.4. Symmetric & Asymmetric Semantic Search

Semantic search is sensitive to a ratio between length of document and length of query, as well as whether query and document can be potentially used interchangeably, that's why semantic search can be symmetric or asymmetric. Symmetric semantic search may consist of query and title of a web article with a similar name, for example for a query "How to install pythorch on M1 Mac?" the good result would be an article called "Everything you need to know about installing pytorch on M1 Mac". Asymmetric semantic search typically consists of a short question-like query (“How to remove a malware?”) and a longer answer answering the question “To remove a malware, try scanning your computer with Defender utility. In case it does not find anything, start your computer in Safe mode …”.

1.2. Classic retrieval approach. BM-25

Due to speed and computational limitations in retrieval problem it is important to understand classic retrieval approaches, as it is still a widespread practice to use them for initial retrieval of a substantial number (typically, top-1000) most relevant documents using faster, simpler algorithms, and then using more advanced, neural based approaches for further re-ranking. BM-25 (Best-Matching 25)[3] is an industry-standard baseline algorithm for initial retrieval task.

BM-25 is based on certain important assumptions about the documents and terms (normalized words) that documents consist of:

1. Words in documents and query are tokenized (split by punctuation characters in most naïve implementation), stemmed (reduced to their roots, i.e., ‘cooker, cooking, cooked -> cook’), lemmatized (reduced verb forms to base form, i.e., \`was, were -> be\`). Stop words (‘in’, ‘at’, etc.) are removed using a pre-defined list.
   Exact implementations for each step may be different.
2. In order to understand term frequencies, extracted terms are used to build an inverted index. Inverted index is a data structure that contains documents metadata and statistics of term frequencies and document frequencies.

3. Terms frequency tf_{td} can be viewed as a table where each document d is represented as a column and each term t as a row. Each table value tf_{td} represents number of times a term t appears in a document d.
   Storing full term-frequency table is inefficient due to the sparsity – thus, inverted index is built – a table, where each term contains a list of documents it appears in and corresponding term frequency in the doc
   $$dog - [d_1: 5, d_{30}: 3]$$
   (Term dog appears five times in doc d_1, three times in doc d_{30})
   Using the raw frequencies is not the best solution, as it would give a popular term too high weight yet produce irrelevant result. (i.e., one doc can contain word "machine" 100 times yet be irrelevant to the query "machine learning"). Logarithm is used to lower the weight of the terms:
   $$tf(t, d) = \log\left(1 + tf_{t,d}\right)$$
4. Document frequency df_t represents how in many documents term t appears. Inverse document frequency is typically defined as:
   $$idf(t) = \log \frac{|D|}{df_t}$$
   Where $|D|$ is a number of documents in a collection, and log is used to get a smaller range of values.
5. To take into account both the term frequency (how often term is represented in a particular document) and document frequency (how many documents contain particular term), TF-IDF score was introduced:
   $$TF_{IDF(d,q)} = \sum_{t \in T_d \cap T_q} tf_{t,d} * idf(t) \; = \sum_{t \in T_d \cap T_q} \log\left(1 + tf_{t,d}\right) \log\left(\frac{|D|}{df_t}\right)$$

Such a function would downgrade scores for frequently used words and give a high score for rare words that appear often in a certain document. Scores of TF-IDF are used as inputs for different NLP and retrieval tasks. The problem of using raw TF-IDF is that despite introduction of logarithm, TF-IDF weights are always increasing with increasing term frequency, which leads to more unbalanced and inaccurate results.

To deal with always increasing weights and provide a function that would saturate with increasing term frequency, BM-25 (ref!) was introduced.

Same as TF-IDF, BM-25 is based on tf and df_t, takes into account document length dl and uses additional hyperparameters k and b that can be optimized for a particular dataset:

$$BM25(q,d) = \sum_{t \in T_d \cap T_q} \frac{tf_{t,d}}{k\left((1-b)+b\frac{dl_d}{avgdl}+tf_{t,d}\right)} * \log\left(\frac{|D|-df_t+0.5}{df_t+0.5}\right)$$

Hyperparameter b is responsible for normalizing document length, and k controls scaling of term frequency.

$1.2 < k < 2$ and $0.5 < b < 0.8$ are frequently used ranges for the hyperparameters.

BM-25 is a basis for most search engines used today.

In our solution, we use BM-25-Anserini implementation[10].

1.3 Evaluation Metrics for re-ranking

Evaluation metrics are another crucial aspect in estimation of the quality of different retrieval systems.

An important concept in re-ranking problem is relevance. It can be assumed that the document is relevant or not relevant to the query (a.k.a., binary relevance). However, a more practical approach would be to assign grades of relevance to a document. This approach is called utility-based relevance and relevance can be represented either as a floating point or a fixed set of classes. Commonly, relevance can be graded as follows:

Text label	Description	Label
Irrelevant	Document does not provide any relevant information related to the query.	0
Relevant	Document provides minimal relevant information related to the query.	1
Highly relevant	Document provides substantial information related to the query.	2
Perfectly relevant	Document is dedicated to the query and can be displayed as a top result in a search engine.	3

Table 1.1 – Common TREC Relevance Labels

Binary relevance can be evaluated with metrics such as MRR and MAP, most important utility relevance metric is NDCG.
As the problem is often setup to re-rank certain subset of top retrieved documents, it is commonly used to notate metrics with cutoff @k, i.e., NDCG@10 means that we calculate the metric only for ten top resulting documents.

1.3.1 MRR@K

Mean Reciprocal Rank, or MRR[11], is a metric that evaluates test results based on where the first relevant item is placed. For a single query,

$$RR = \frac{1}{rank}$$

where $rank$ is a position of highest ranked answer, ranged from 1 to N, where N is the number of results for a given query. For multiple queries Q,

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$$

MRR assumes that the user is interested only in the first most relevant document, which may be true in applications and scenarios.

1.3.2 MAP@K

Mean Average Precision, or MAP[12], is a metric that evaluates retrieval system based on an assumption that the user is interested in each relevant document and is willing to judge the system based on how it performs on multiple queries.

Firstly, let's remind what precision metric stands for. Given classification task with lists of binary labels and predictions (0s and 1s, negatives and positives), we define precision as a share of the correctly identified 1s among all items labeled as 1.

$$\mathrm{Pr} = \frac{tp}{tp + fn}$$

Where tp stand for positive labels that were matched as positive, fn stand for positive labels that were matched as negative.

Average precision can be defined as follows:

$$AP(q) = \frac{\sum_{i=1}^{k} P(q)_{@i} * rel(q)_i}{|rel(q)|}$$

where q is a given query, $P(q)_{@i}$ is a precision of q after the first i documents, $rel(q)_i$ stands for binary relevance of doc at position i, and $|rel(q)|$ is a total number of relevant documents for the query.

MAP means AP over multiple queries:

$$MAP(Q) = \frac{1}{|Q|} \sum_{q \in Q} AP(q)$$

MAP gives more focus to errors associated with wrong documents that appear high in the re-ranked list. Its main downside lies in inability to work with non-binary relevance ratings.

1.3.3 NDCG@K

Non-discounted cumulative gain (NDCG)[13] also has emphasis on correctness of documents high in the ranking list. An idea behind NDCG is that the most relevant items should be first, followed by somewhat relevant items and the least relevant items should be last in re-ranked list of documents.

Given that each document has ranked position $i, i \in [1; n]$ and relevance value $rel(d)$ (typically, 0-3), Discounted Cumulative Gain (DCG) can be defined as follows:

$$DCG(D) = \sum_{d \in D} \frac{rel(d)}{\log_2(i+1)}$$

Given a set of queries Q, we can define $nDCG$:

$$nDCG(Q) = \frac{1}{|Q|} \sum_{q \in Q} \frac{DCG(q)}{DCG(sorted(rel(q))}$$

Where $sorted\big(rel(q)\big)$ stands for the best documents ranking possible.

NDCG has advantage over MRR, as it gives less discounts to documents that appear lower in ranking list.

2. Using Language Models in semantic search, re-ranking and retrieval. GPT-3 and OPT

Pre-trained language models built using transformer-based architecture[1] caused a paradigm shift in Natural Language Processing. Instead of training separate model from scratch for each task, industry moved towards an idea that every task can potentially be solved with a “universal” language model. In order to prepare such a model, one has to train a model that learns representations for words and sentences, with similar words being “located nearby” and opposite words being “far away from each other”.

This concept is called word embeddings and it implies that each word can be encoded as a real-value vector in N-dimensional space.

The first architecture under the paradigm of pre-training was BERT[2]. Its success increased effort towards training a single transformer-based model capable of solving NLP tasks in few-shot, or even zero-shot manner, and influenced the appearance of GPT-3[5], that showed phenomenal results and revolutionized the field even further.

Re-ranking problem also benefited from appearance of pre-trained language models. Quality vector representations for queries and documents led to introduction of different new solutions [7], [14] that beat previous approaches by a margin.

Once GPT-3 appeared, it demonstrated huge gains over BERT in a variety of tasks. Re-ranking is no exception, and as it was demonstrated[6], GPT- has huge potential in re-ranking task. As more open-source GPT-3 implementations are appearing, the focus of this paper is on one of them, OPT[8]. OPT is an implementation of GPT-3 with minor architectural changes. OPT models vary in size, from 125M to 175B.

2.1 GPT network architecture and related concepts

Language modelling is a machine learning task of predicting words given a text sequence. GPT presents very effective architecture for this task. As for input, it operates on parts of words, known as tokens. Input layer consists of token embeddings (vocabulary size, hidden dim) and positional encoding vector that explains the model words position in the texts[15].

A main component of GPT is a decoder block, that consists of masked self-attention, followed by feed forward neural network. The concept behind self-attention lies in constructing a probability distribution for each token in a particular sequence given its position. Queries, Keys and Value vectors are a basis of self-attention. Query vector represents a given token in order to score it against other tokens in a sequence, Keys vectors represents other tokens, and Values correspond to word representations. Queries and Key vectors are then multiplied and summed in order to get vector of scores of what can be interpreted as a "correlation matrix" between words.

In order to treat long sequences more effectively, separate parts of keys-queries-values vectors are split, scored separately and gathered as separate attention-heads. Merged attention heads are then passed towards a projection layer so that it could be processed by a feed-forward layer in order to get output of the input shape and feed it to the next decoder block.

2.2 Interaction with GPT. Prompts

The main advantage of pre-trained GPT models is the ease of interaction. Once model is trained, user different NLP tasks can be solved by feeding text inputs into the network. In order to make the model "solve" a particular NLP task, a correct input pattern must be designed. Input patterns are called prompts, and the process of selecting a correct pattern is called prompt engineering. Prompts are divided into zero-shot and few-shot. Zero-shot means that the model is not provided with any examples, and few-shot means that we give some examples during inference step. For example, a translation task may be approached in zero-shot manner via prompt like:

$$Translate\ text\ from\ English\ to\ French\ \backslash n\ text{:} < text >\backslash ntranslation{:}$$

Few-shot prompt may look like:

$$Translate\ English\ to\ French{:}\ cheese \rightarrow fromage\backslash n\ wine \rightarrow vin\backslash n\ bread \rightarrow$$

To fit a model to a new task or to make it work in zero-shot for a particular prompt model may be trained further using some new data. This process is called fine-tuning and is a more traditional approach of adopting language models for new tasks.

2.3. GPT and re-ranking. Zero-shot re-ranking solution

Prompt-engineering is the easiest, "naïve" way to use GPT in re-ranking task. A prompt can be any string that contains slots for document and query, prompt options are covered in Section 3.2.

Given a query and a set of documents, the most suitable document to the query is:

$$d^* = \underset{d \in D}{\operatorname{argmax}} P(d|q)$$

Using Bayes theorem,

$$d^* = \underset{d \in D}{\operatorname{argmax}} P(d|q) = \underset{d \in D}{\operatorname{argmax}} \frac{P(q|d)}{P(q)} = \underset{d \in D}{\operatorname{argmax}} P(q|d)P(d)$$

Thus, $P(q|d)$ can estimate how "good" a query fits a document. As GPT is auto regressive, it can predict $P(q|d)$ by placing d to the left part of the prompt and q to the right part of it. Given tokenized input sequence (prompt) p of length

$n, p = (p_{1,\dots,i-1}, q_{i,\dots,n})$ with query tokens $q_{i,\dots,n}$ and prompt tokens (including document) $p_{1,\dots,i-1}$,

$$score(d) = P(q|d) = \sum_{k=i}^{n} \log\left(softmax(q_k)\right)$$

Documents can be sorted by the calculated scores.

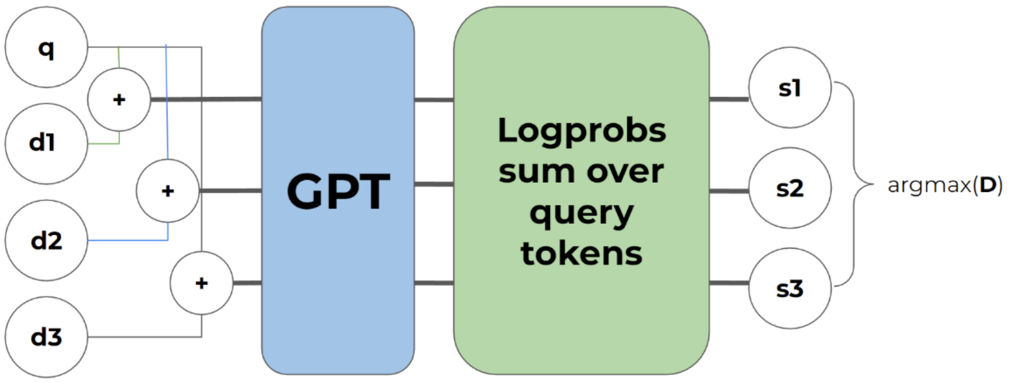

Figure 1 – Naïve approach for GPT-based re-ranking

2.4 Cross-Encoder and Bi-Encoder models

Scoring approach described in Section 2.3 may be useful for offline usage or quick prototyping but has its own limitations in actual retrieval and re-ranking pipeline. Its performance may be increased, its main disadvantage is slow speed and number of comparisons required to rank multiple query-documents sets.

Like other transformers used for the re-ranking task, GPT out of the box performance can be improved via making minor architecture changes and fine-tuning slightly changed model on datasets of document-query pairs.

2.4.1 Bi-Encoder

Bi-Encoder approach is based on an intuitive idea to cover the whole retrieval and re-rank pipeline by calculating similarity between vector representations (embeddings) of documents and queries. Using document representation as a vector is useful for faster retrieval, as document vectors may be indexed and compared against queries in a real-time setting

The first step in Bi-Encoder setting is to get a single embedding vector for the input query/document, that would have same dimensionality regardless of the sequence length. In order to achieve that, Bi-Encoder uses pooling operation between LM output vectors for each token in an input sequence. Based on the architecture of the used LM, different types of pooling may be used.

Given document and query embeddings, one can rank documents by using cosine similarity.

In order to get better representations, Bi-Encoder may be fine-tuned, as described in [6][16].

2.4.2 Cross-Encoder

Cross-Encoder is a classifier or regressor built on top of a language model. Given a query-document pair, it calculates a relevance score (usually, in range [0,1]. Before usage, Cross-Encoder is fine-tuned on datasets of triplets (query, document, relevancy), where relevancy is normalized to a desired score range.

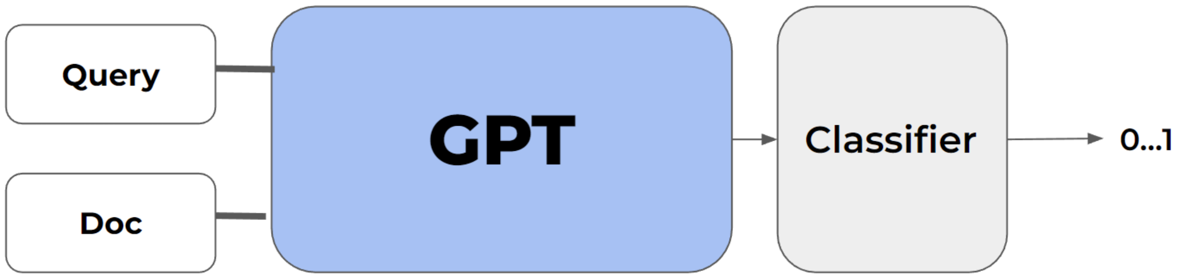

Figure 2 Cross-Encoder architecture

Cross-Encoder is the most accurate existing method for re-ranking. Its main downside, however, is that it doesn't store vector representations, and is not fast enough to score millions of document-query pairs in real time.

Search systems that rely on neural networks only typically use Bi-Encoder for retrieval step (as document representations are already stored) and Cross-Encoder for re-ranking step on a narrow subset of pre-selected documents.

3. Practical research

Research part of the paper is focused on OPT[8]. This paper presents first OPT evaluation on BEIR benchmark[9] using different prompts, as well as trains first GPT-3 based Cross-Encoder, also using OPT. The OPT Cross-Encoder (OPT-125M) outperforms classical approach (BM-25), previous-generation SOTA[7] as well as out-of-the box Naïve method of re-ranking. The last part of the research is focused on improving performance of smaller OPT-model (OPT-125M) using various knowledge distillation techniques, such as self-attention distribution transfer.

3.1. Training dataset: MS MARCO Passage

Microsoft Machine Reading Comprehension is a dataset focused on passage ranking, question answering and machine reading comprehension. Introduced by Microsoft as a part of TREC-2019 challenge, it consists of 8.81million query-document pairs.

During experiments, a subset of 2 million query-document pairs from train set were used for fine-tuning experiments, and eval subset was used as a part of BEIR benchmark.

3.2 Benchmark for retrieval and re-ranking: BEIR

In order to compare quality of different retrieval systems it requires evaluating them on different tasks and benchmarks. Evaluation in IR can be done offline or online, where online evaluation is done by doing A/B tests and getting feedback from the users, while offline evaluation can be done using

fixed test collections of queries (can be handcrafted or sampled from real-world user queries), documents (task-specific) and judgements about query-document relevance.
Most commonly, re-ranking benchmarks are datasets from TREC[17] and NTCIR[18] test collections, which are updated yearly by the community of the two largest conferences related to Information Retrieval.

However, this paper uses a recently developed benchmark called BEIR (Benchmarking IR). BEIR is an evaluation benchmark that consists of 19 publicly available datasets from 10 different text retrieval tasks. [add reference!]. BEIR was specifically designed to introduce more accurate representation of real-world usage for re-ranking models and is now widely used to benchmark different re-ranking approaches and models[6]. Apart from being set in different domains, such as BIO-Medical IR, Question Answering, Tweets-Retrieval, Fact Checking, Citation Prediction, etc., BEIR datasets differ in number of test document-query pairs, document-query ratio, and, most importantly, use different grades of relevancy, which allows researchers to identify possible shortcomings in particular model during training and evaluation stages.

Note that in this work we didn't use Signal-1M, TREC-News, Bioasq and Robust-04 datasets, as they required separate enquiries for access. That is why benchmarked is marked as BEIR* in results section.

The following table provides a short statistic of datasets presented in BEIR benchmarks:

Task	**Dataset**	**Relevancy**	**Test Queries**	**Test Docs**
Passage-Retrieval	MS Marco	Binary	6980	8841823
Bio-Medical IR	TREC-COVID	3-level	50	171332
Bio-Medical IR	NFCorpus	3-level	323	3633
Bio-Medical IR	BioASQ	Binary	500	14914602
Question Answering	NQ	Binary	3452	2681468
Question Answering	HotPotQA	Binary	7405	5233329
Question Answering	FiQA-2018	Binary	648	57638
Tweets-Retrieval	Signal-1M (RT)	3-level	97	2866316
News Retrieval	TREC-NEWS	5-level	57	594977
News Retrieval	Robust04	3-level	249	528155
Argument Retrieval	ArguAna	Binary	1406	8674
Argument Retrieval	Touché-2020	3-level	49	382545
Duplicate-Question Retrieval	CQADupStack	Binary	13145	457199
Duplicate-Question Retrieval	Quora	Binary	10000	522931
Entity-Retrieval	DBPedia	3-level	400	4635922
Citation-Prediction	SCIDOCS	Binary	1000	25657
Fact Checking	FEVER	Binary	6666	5416568
Fact Checking	Climate-FEVER	Binary	1535	5416593
Fact Checking	SciFact	Binary	300	5183

Table 3.1 - BEIR datasets statistics

3.2. Prompt Engineering and OPT evaluation

In order to use OPT on re-ranking task, we had to start with selecting the most suitable prompt that would unlock OPT potential and show good results.

Based on [6], three prompts were chosen for evaluation OPT on BEIR dataset using method described in Section 2.3. Additionally, GPT-NEO [19] was also evaluated in order to compare performance differences between different GPT implementations.

Selected prompts were:

Number	**Text**
Prompt 1	Documents are searched to find matches with the same content.\nThe document "{doc}" is a good search result for "{query}
Prompt 2	Documents are searched to find matches with the same content.\nDocument: "{doc}"\n\nThe above document is a good match for the query: "{query}
Prompt 3	The selected text is:\n{doc}\n\n\nThe relevant title is:\n{query}

Table 3.2 – Prompts list

The table below illustrates used prompts and their respective mean BEIR* NDCG@10 scores:

Prompt	**Network**	**NDCG@10**
Prompt 1	OPT-125M	0.3998
Prompt 2	OPT-125M	**0.4084**
Prompt 3	OPT-125M	0.4065
Prompt 1	OPT-350M	0.4098
Prompt 2	OPT-350M	**0.4137**
Prompt 3	OPT-350M	0.4116
Prompt 1	GPT-NEO-125M	0.3137
Prompt 2	GPT-NEO-125M	**0.3153**
Prompt 3	GPT-NEO-125M	0.3102

Table 3.3 – Zero-shot evaluation results

As may be seen, OPT-125M beat GPT-NEO-125M out of the box with all three prompts. OPT-350M slightly outperforms OPT-125M and prompts 1 and 3 showed opposite results in OPT to those under GPT-NEO. Prompt 2 showed better results among three options; thus, further experiments were done using it for input sequences.

3.3. Training a Cross-Encoder

Easiest way to improve OPT performance out of the box was to train Cross-Encoder. In order to do so, we used Prompt-1. Architecturally, a linear layer with a sigmoid activation function was placed on top of the output GPT layer. Input dim of linear layer equals hidden dim of the model (768 for OPT-125M, 1024 for OPT-350M). Cross-Encoder was trained on data described in Section 3.1 and Mean-Squared Error was used as a loss function. AdamW was used as an optimizer, with learning rate set to 1e-5 and weight decay of 0.001. Fine-tune took exactly one epoch and improved OPT-125M performance by ~2%. Also, GPT-NEO-125M cross-encoder was trained, but it showed worse results that out-of-the box model, which requires separate investigation.

3.3. Improving OPT performance: Knowledge Distillation

Another way to improve performance of OPT-125M is to fine-tune it in a way that it would mimic a behavior of bigger models. This technique is called Knowledge distillation. Under this setting, one model is trained using representations or labels generated by a more capable model. The model that is being trained is called a student, and a more powerful model that student learns to imitate is called a teacher. The motivation behind these experiments lies in an idea to create an OPT version of mini-LM-v2, a BERT-based SOTA of fast yet highly capable retrieval & re-rank model.

Training data and optimizer settings were used the same as in Cross-Encoder experiments. Knowledge distillation experiments were performed using OPT-350M as a teacher and OPT-125M as a student. Kullback-Leibler divergence [20] was used as a loss function.

3.3.1. Logits distribution transfer

The simplest way to perform knowledge distillation is to make student model imitate logits distribution of teacher model. In order to do so, KL-divergence between student and teacher models is used as a loss function for a student:

$$L = D_{KL}(\log(softmax(logits_T)||\log(softmax(logits_S))$$

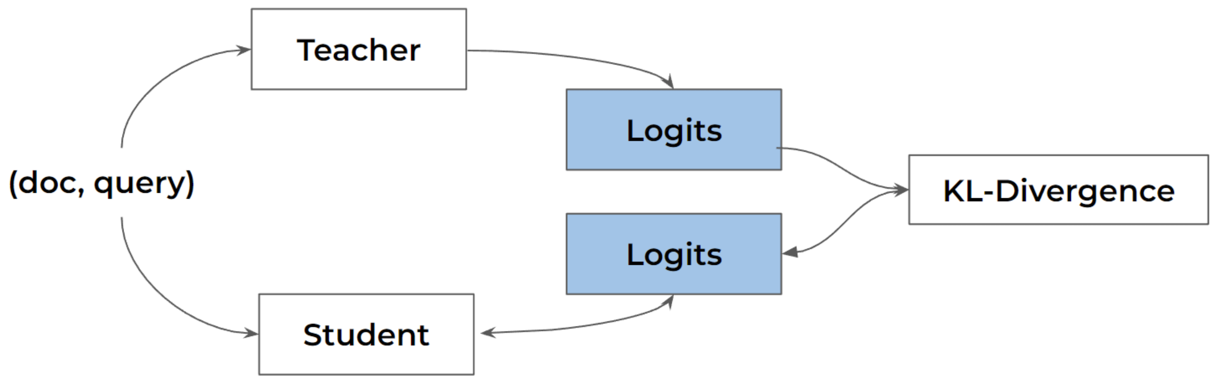

Figure 2

3.3.2. Self-attention relation distillation

A more complex approach is to make a model imitate self-attention mechanism. The selected method is called self-attention relation distillation and was successfully used in [7]. As our models do not differ in size that greatly, we used outputs from corresponding last attention layers of student and teacher models. Loss function can be defined:

$$L = L_Q + L_K + L_V$$

Where $\forall W \in [Q, K, V]$ L_W is a KL divergence between (K-K, V-V or Q-Q) scaled dot-product matrices, which represent relations between different parts of attention mechanism.

$$L_W = \frac{1}{A_r|x|}\sum_{a=1}^{A_r}\sum_{t=1}^{x} D_{KL}(R_{a,t}^{T}||R_{a,t}^{S})$$

Where $|x|$ stands for sequence length, A_r – number of student attention heads, D_{KL} is a KL-divergence,l, m mean and

$$R_{a,t}^{X} = softmax(\frac{A_a^X A_a^X}{\sqrt{d_r}})$$

is a self-attention relation between (K-K, V-V or Q-Q) of teacher (T) or student (S) model. d_r represents number of student attention heads.

(K-K, V-V, Q-Q) are multi-head relation vectors that are aggregated between attention heads, concatenated, split if number of teacher model attention heads is higher than student attention head, and then multiplied via dot product.

3.4. Results comparison table

	NDCG*@1	NDCG*@10	NDCG*@100
Naïve Baseline (BM-25)	0.2175	0.228	0.2816
Transformer SOTA (*cross-encoder/ms-marco-MiniLM-L12-v2*)	0.3317	0.3703	0.4041
GPT-NEO-125M, zero-shot	0.2925	0.3153	0.3713
OPT-125M, zero-shot	0.3751	0.4084	0.4408
OPT-125M Cross-Encoder	**0.3915**	**0.4233**	**0.4619**
OPT-125M-distilled-logits-transfer	0.3516	0.3772	0.3819
OPT-125M-distilled-attention-transfer	0.3867	0.4106	0.4315
OPT-350M, zero-shot	0.3911	0.4137	0.4578

Table 3.4 – summary comparison of experiment results

As mentioned before, results are measured on a subsection of BEIR. Evaluation metric is NDCG, NCDG* stands for mean results over present datasets.

3.5. Results

OPT evaluation demonstrated that it outperforms its GPT-NEO competitor by a huge margin with all experimented prompts. Moreover, with a correct prompt it outperforms BERT-based SOTA solution out of the box.

In terms of model sizes, there is little difference between OPT-125M and OPT-350M in zero-shot performance and using OPT-350M as a teacher doesn't add a lot of performance to OPT-125M.

Logits transfer experiment failed and degraded model performance, while self-attention relation transfer boosted out-of-the box performance, which lets a room for improvement and further experiments in order to find better knowledge-distillation setting. It is worth trying two-step distillation, from 1.3B model to 350M and from 350M to 125M.

Training a Cross-Encoder was the most successful experiment, as it easily outperformed all the competitive solutions.

It is worth noting that results are heavily reliant on prompts, and a different prompt may heavily influence the resulting benchmark numbers.

Overall, all neural-based approaches outperform BM-25 baseline, though, may be much slower.

Conclusion

In this work we have discusses semantic search tasks based on neural networks of GPT-3 architecture. The work introduced fresh GPT-3 implementation, OPT, to the search problem of re-ranking. In the work, we chose the most suitable input prompt for re-ranking with OPT by evaluating OPT on BEIR benchmark in zero-shot mode with different input prompts. The best proposed prompt let OPT outperform both its predecessor (sBERT) and other GPT-3 implementation (GPT-Neo) by a significant margin, which showed its usability for re-ranking out of the box. In order to improve OPT zero-shot performance in re-ranking for smaller OPT model we have conducted experiments under different knowledge distillation settings by using logits distribution transfer and self-attention relation distillation. The experiments were rather successful, as distilled smaller model showed similar performance to the larger model. Inspired by OPT-results in zero-shot, the work proposed OPT-based Cross-Encoder model that outperforms SOTA Cross-Encoders. Obtained results demonstrate utility of using large pre-trained language models in tasks related to neural search and confirm the validity and prospects of using GPT in re-ranking task.

Further work would involve building an OPT model that could maintain re-ranking quality of larger OPT models, while surpassing them in faster inference speed and smaller model size.

References

1. Vaswani, A., et al. Attention Is All You Need, 2017, doi:10.48550/arXiv.1706.03762
2. Devlinn, J., et al: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018, doi:10.48550/arXiv.1810.04805
3. Robertson, S., Zaragoza, H., The Probabilistic Relevance Framework: BM-25 and Beyond, 2009, doi: 10.1561/1500000019
4. Humeau,S., et al. Poly-encoders: Architectures and Pre-training Strategies for Fast and Accurate Multi-sentence Scoring ,2020, doi:10.48550/arXiv.1905.01969
5. Brown, T.B., et al. Language Models are Few-Shot Learners, 2020, doi:10.48550/arXiv.2005.14165
6. Muennighoff, Niklas. SGPT: GPT Sentence Embeddings for Semantic Search, 2022, doi:10.48550/arXiv.2202.08904
7. Wang, W., et al. MiniLMv2: Multi-Head Self-Attention Relation Distillation for Compressing Pretrained Transformers, 2021, doi:10.48550/arxiv.2012.15828
8. Zhang, S., et al. OPT: Open Pre-Trained Transformer Language Models, 2022, doi:10.48550/arXiv.2205.01068
9. Thakur, N., et al. BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models, 2021, doi:10.48550/arXiv.2104.08663
10. Lin, J., et al. Pyserini: An Easy-to-Use Python Toolkit to Support Replicable IR Research with Sparse and Dense Representations, 2021, doi:10.48550/arXiv.2102.10073

11. Voorhees, Ellen M. The trec-8 question answering track report. Trec. Vol. 99, 1999.
12. Sanderson, M., Manning C.D., et al. Introduction to Information Retrieval, Cambridge University Press. 2008. ISBN-13 978-0-521-86571-5, xxi+ 482 pages.
13. Järvelin, Kalervo, and Jaana Kekäläinen. "Cumulated gain-based evaluation of IR techniques." ACM Transactions on Information Systems (TOIS) 20.4 (2002): 422-446.
14. Reiemers, Nis, Gurevych, Iryna, Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks, 2019, doi:10.48550/arXiv.1908.10084
15. Radford, A., et. al, Language Models are Unsupervised Multitask Learners, 2019, URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf
16. Lin, J., et al., Pretrained Transformers for Texts Ranking: BERT and Beyond , 2021, doi:10.48550/arXiv.2010.06467
17. http://trec.nist.gov/
18. https://research.nii.ac.jp/ntcir/index-en.html
19. Sid, B., et al. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, doi:10.5281/zenodo.5297715
20. Kullback, S. and Leibler, R.A. On information and sufficiency. The Annals of Mathematical Statistics, 22, 79-86. 1951, doi:10.1214/aoms.1177729694