Transformers in Information Retrieval

[DAT640] Information Retrieval and Text Mining

Petra Galuscakova
University of Stavanger

September 18, 2024

In this module

1. Multi-Stage Architectures for Reranking
2. Refining Query and Document Representations
3. Learned Dense Representations for Ranking
4. Retrieval-Augmented Generation

Exercise

E8-1 Designing IR System with Transformers Exercise

Multi-Stage Architectures for Reranking

Neural Ranking

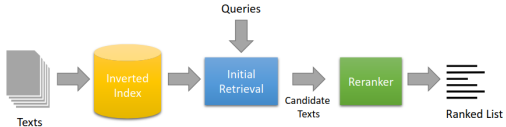
- The simplest and most straightforward formulation of text ranking is to convert the task into a **text classification** problem, and then sort the texts to be ranked based on the probability that each item belongs to the desired class (relevant/irreleant)
- The approach involves training a classifier to estimate the probability that each text belongs to the 'relevant' class, and then at ranking (i.e., inference) time sort the texts by those estimates.

Multi-stage Ranking

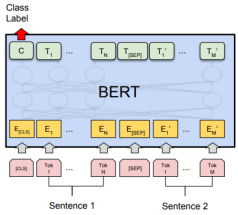
- A system could have an arbitrary number of reranking stages, where the output of each reranker feeds the input to the next.
- We formalize the design as follows: a multi-stage ranking architecture comprises N reranking stages, denoted H_1 to H_N .
- We refer to the candidate generation stage (initial retrieval or first-stage retrieval) as H_0, which retrieves k_0 texts from the corpus to feed the rerankers.
- Each stage H_n, $n \in 1, \ldots N$ receives a ranked list $R_{n\text{-}1}$ comprising $k_{n\text{-}1}$ candidates from the previous stage and, in turn, provides a ranked list R_n comprising k_n candidates to the subsequent stage
- One practical motivation for the development of multi-stage ranking is to better balance tradeoffs between effectiveness (most of the time, referring to the quality of the ranked lists) and efficiency (e.g. retrieval latency or query throughput).

monoBERT

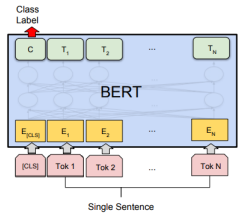
- The task of relevance classification is to estimate a score s_i quantifying how relevant a candidate text d_i is to a query q, which we denote as: $P(Relevant = 1|d_i, q)$.
- It is impractical to apply BERT inference to a large number of texts for every query.
- Simple and fast retrieval model (usually BM25) is first applied to create list of candidate relevant documents
- BERT inference is then applied to rerank these candidates to generate a score s_i for each text d_i in the candidates list. The BERT-derived scores may or may not be further combined or aggregated with other relevance signals to acquire at the final scores used for reranking.



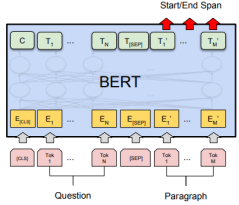
BERT model - applications



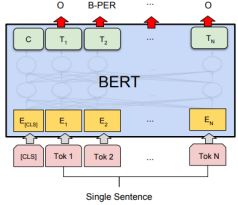
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA



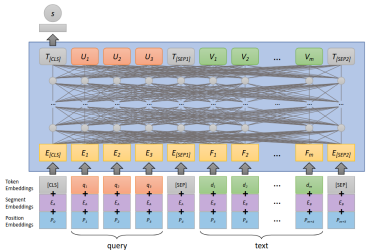
(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Basic Design of monoBERT

- For the relevance classification task, the model takes as input a following sequence: $[[CLS], q, [SEP], d_i, [SEP]]$, where q comprises the query tokens and d_i comprises tokens from the candidate text to be scored.
- This general style of organizing task inputs (query and candidate texts) into an input template to feed to a transformer for inference is called a 'cross-encoder'.
- It stores lot of complex information about interaction, but it needs to encode every document for each new query, what is slow.



monoBERT Effectivness

Figure 8: The effectiveness of monoBERT$_{\text{Base}}$ on the development set of the MS MARCO passage ranking test collection varying the amount of training data used to fine-tune the model and reranking $k = 1000$ candidate texts provided by first-stage retrieval using BM25. Results report means and 95% confidence intervals over five trials.

Reranking Lists of Texts

- There are multiple scoring functions to optimally rank the documents given a query:
  - Pointwise: Predict the absolute relevance (for each item in the list, predict the similarity score) – such as in the case of monoBERT
  - Pairwise: Predict the ranking of a document pair (predict the ranking between query and pairs of documents)
  - Listwise: Predict the ranking of a document list (predict the ranking directly)

Reranking Pairs of Texts

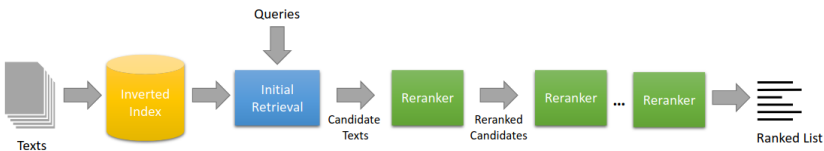
- The duoBERT model uses pairwise ranking and considers pairs of text. In this ranking model, BERT is trained to estimate the following: $P(d_i \succ d_j | d_i, d_j, q)$, where $d_i \succ d_j$ is stating that d_i is more relevant than d_j (with respect to the query q).
- It takes as input a sequence comprised of a query and two texts, comprising the input template: [[CLS], q, [SEP], d_i, [SEP], d_j, [SEP]]
- Due to the length limitations of BERT, the query, candidates d_i and d_j are truncated to 62, 223, and 223 tokens, respectively,

duoBERT cont.

- At inference time, the pairwise scores $p_{i,j}$ are aggregated so that each document receives a single score s_i.
- Most frequently it is done using a SUM: $s_i = \sum_{j \in J_i} p_{i,j}$ where $J_i = \{0 \leq j < |D|, j \neq i\}$ is used
- Other functions used include minimum, maximum, or binary comparison

mono-duoBERT

- Typical architecture:
  - BM25 ranker in first stage
  - monoBERT (k=1000) in the second stage
  - duoBERT (k=50) in the third stage



mono-duoBERT Performance

	Method	MS MARCO Passage Development MRR@10	MS MARCO Passage Test MRR@10
(1)	Anserini BM25 = Table 5, row (3a)	0.187	0.190
(2)	+ monoBERT ($k_0 = 1000$) = Table 5, row (3b)	0.372	0.365
	+ monoBERT ($k_0 = 1000$)		
(3a)	+ duoBERT$_{\text{MAX}}$ ($k_1 = 50$)	0.326	-
(3b)	+ duoBERT$_{\text{MIN}}$ ($k_1 = 50$)	0.379	-
(3c)	+ duoBERT$_{\text{SUM}}$ ($k_1 = 50$)	0.382	0.370
(3d)	+ duoBERT$_{\text{BINARY}}$ ($k_1 = 50$)	0.383	-
(4a)	+ monoBERT + TCP	0.379	-
(4b)	+ monoBERT + duoBERT$_{\text{SUM}}$ + TCP	0.390	0.379

Table 21: The effectiveness of the monoBERT/duoBERT pipeline on the MS MARCO passage ranking test collection. TCP refers to target corpus pretraining.

Pre-trained BERT for IR

- The BERT model used in IR is usually pre-trained on the MS MARCO collection and then potentially fine-tuned on the target collection
- MS MARCO passage ranking test collection: 8.8 million pairs of query and relevant passages
- The collection is organized for training – i.e. it contains triples: query, relevant passage, irrelevant passage
- Pre-trained models on MS MARCO can be found on Huggingface: e.g. https://huggingface.co/cross-encoder/ms-marco-TinyBERT-L-2
- Updated version (v2), collection of documents also exists
- Multilingual MS MARCO models also exist and were trained on translated MS MARCO

Document Ranking

- The maximum sequence length of 512 tokens presents a challenge to using BERT for ranking longer texts. There are multiple strategies how to deal with longer texts.
- BERT–MaxP:
  - For training: Segment documents into overlapping passages. Treat all segments from a relevant document as relevant and all segments from a non-relevant document as not relevant.
  - For the inference: segment documents in the same way, estimate the relevance of each passage, and then perform simple aggregation of the passage relevance scores (e.g. taking a maximum) to acquire the document relevance score.

Passage Score Aggregation

- BERT–MaxP: take the maximum passage score as the document score, i.e., $s_d = max(s_i)$
- BERT–FirstP: take the score of the first passage as the document score, i.e., $s_d = s_1$.
- BERT–SumP: take the sum of all passage scores as the document score, i.e., $s_d = \sum_i s_i$.
- Comparing the different aggregation techniques, the MaxP approach appears to yield the highest effectiveness in many cases

T5 Model in Mono and Duo Setup

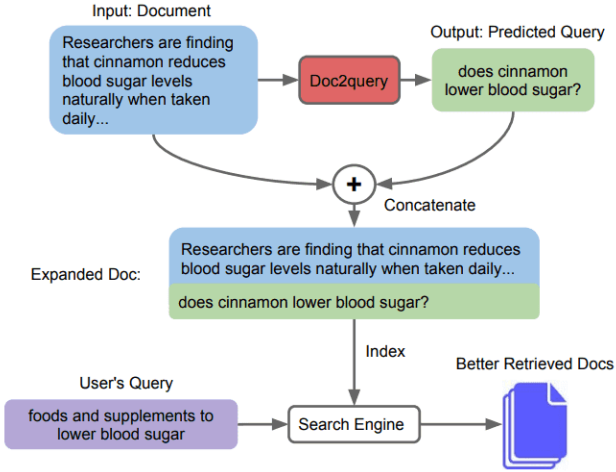
- Mono and Duo architectures can also be used with encoder-decoder architectures such as T5
- MonoT5: uses a following template: query: [q] Document: [d] Relevant:
  where [q] and [d] are replaced with the query and document texts, respectively, and the other parts of the template are verbatim string literals.
- The model is fine-tuned to produce the tokens 'true' or 'false' depending on whether the document is relevant or not to the query.
- At inference time, to compute probabilities for each query–text pair, a softmax is applied only to the logits of the 'true' and 'false tokens
- monoT5-base has a higher effectiveness than monoBERT-large
- DuoT5 uses a following template: Query: q Document0: di Document1: dj Relevant:

Refining Query and Document Representations

Query and Document Expansion

- Query expansion and document expansion techniques provide two potential solutions to the vocabulary mismatch problem.
- The basic idea behind document expansion is to augment (i.e., expand) texts from the corpus with additional terms that are representative of their contents or with query terms for which those texts might be relevant.

Document Expansion via Query Prediction: doc2query



Document Expansion via Query Prediction: doc2query

- The basic idea is to train a sequence-to-sequence model that, given a text from a corpus, produces queries for which that document might be relevant.
- This can be thought of as 'predictively' annotating a piece of text with relevant queries.
- Given a dataset of (query, relevant text) pairs, a sequence-to-sequence model can be trained to generate a query given a text from the corpus as input.
- The predictions are appended to the original texts from the corpus without any special markup to distinguish the original text from the expanded text, forming the expanded document.
- This expansion is performed on every text from the corpus, and the results are indexed as usual. The resulting index can then provide a drop-in replacement compatible with any of the ranking/reranking models.

Examples of doc2query

Input: July is the hottest month in Washington DC with an average temperature of 27°C (80°F) and the coldest is January at 4°C (38°F) with the most daily sunshine hours at 9 in July. The wettest month is May with an average of 100mm of rain.

Target query: what is the temperature in washington

doc2query–base: weather in washington dc
doc2query–T5: what is the weather in washington dc

Input: The Delaware River flows through Philadelphia into the Delaware Bay. It flows through and *(sic)* aqueduct in the Roundout Reservoir and then flows through Philadelphia and New Jersey before emptying into the Delaware Bay.

Target query: where does the delaware river start and end

doc2query–base: what river flows through delaware
doc2query–T5: where does the delaware river go

Input: sex chromosome - (genetics) a chromosome that determines the sex of an individual; mammals normally have two sex chromosomes chromosome - a threadlike strand of DNA in the cell nucleus that carries the genes in a linear order; humans have 22 chromosome pairs plus two sex chromosomes.

Target Query: which chromosome controls sex characteristics

doc2query–base: definition sex chromosomes
doc2query–T5: what determines sex of someone

Term Reweighting as Regression: DeepCT

- The term frequency (i.e., the number of times the term appears in a particular text) is the primary feature that attempts to capture the term's importance in the text.

	Term weight: 0.0 0.1 0.2 0.3 0.4 ≥ 0.5
Query	**who is susan boyle**
Relevant	Amateur vocalist **Susan Boyle** became an overnight sensation after appearing on the first round of 2009's popular U.K. reality show Britain's Got Talent.
Non-Relevant	Best Answer: a troll is generally someone who tries to get attention by posting things everyone will disagree, like going to a **susan boyle** fan page and writing **susan boyle** is ugly on the wall. they are usually 14-16 year olds who crave attention.
Query	**what values do zoos serve**
Relevant	**Zoos serve** several purposes depending on who you ask. 1) Park/Garden: Some **zoos** are similar to a botanical garden or city park. They give people living in crowded, noisy cities a place to walk through a beautiful, well maintained outdoor area. The animal exhibits create interesting scenery and make for a fun excursion.
Non-Relevant	There are NO purebred Bengal tigers in the U.S. The only purebred tigers in the U.S. are in AZA **zoos** and include 133 Amur (AKA Siberian), 73 Sumatran and 50 Malayan tigers in the Species Survival Plan. All other U.S. captive tigers are inbred and cross bred and do not **serve** any conservation **value**.
Query	**do atoms make up dna**
Relevant	**DNA** only has 5 different **atoms** - carbon, hydrogen, oxygen, nitrogen and phosphorous. According to one estimation, there are about 204 billion **atoms** in each **DNA**.
Non-Relevant	Genomics in Theory and Practice. What is Genomics. Genomics is a study of the genomes of organisms. It main task is to determine the entire sequence of **DNA** or the composition of the **atoms** that make up the **DNA** and the chemical bonds between the **DNA atoms**.

Term Reweighting as Regression: DeepCT

- What if we were able to directly estimate the importance of a term in the context that the term appears in?
- DeepCT uses a BERT-based model that receives as input a text d and outputs an importance score $y_{t,d}$ for each term t in d.
- The goal is to assign high scores to terms that are central to the text, and low scores to less important terms.
- Like doc2query, DeepCT is trained using (query, relevant text) pairs from the MS MARCO passage ranking test collection.

Term Reweighting as Regression: DeepCT cont.

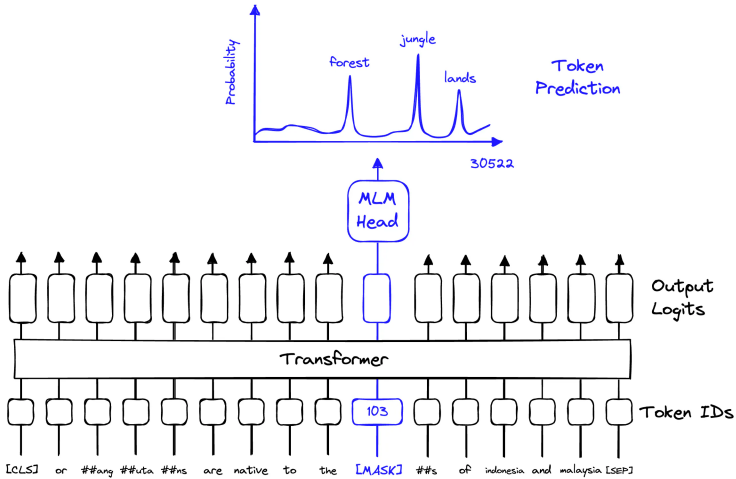
- Once the regression model has been trained, inference is applied to compute $y_{t,d}$ for each text d from the corpus. These weights are then rescaled from [0..1] to integers between 0 and 100 so they resemble term frequencies in standard bag-of-words retrieval methods.
- Finally, the texts are indexed using these rescaled term weights
- New 'pseudo-documents' are created in which terms are repeated the same number of times as their importance weights.
- For example, if the term 'boyle' is assigned a weight of four, it is repeated four times, becoming 'boyle boyle boyle boyle' in this new pseudo-document.
- A new corpus comprising these pseudo-documents, in which the repeated terms are concatenated together, is then indexed like any other corpus. Retrieval is performed on this index as with any other bag-of-words query.

Learned Dense Representations for Ranking

The Sparse Lexical and Expansion model: SPLADE

- Pretrained language model like BERT can identify connections between words/sub-words and use that knowledge to enhance our sparse vector embedding.
- This works in two ways, it allows us to weigh the relevance of different terms ('the' will carry less relevance than a less common word like 'orangutan'). And it enables term expansion: the inclusion of alternative but relevant terms beyond those found in the original sequence.
- BERT and the MLM head are then optimized for predicting the original word/sub-word token that had been replaced by a [MASK] token.
- The MLM head contains 30522 output values for each token position. These 30522 values represent the BERT vocabulary and act as a probability distribution over the vocab.
- These 30522 probability distributions act as an indicator of which words/tokens from the vocabulary are most important for the masked word.

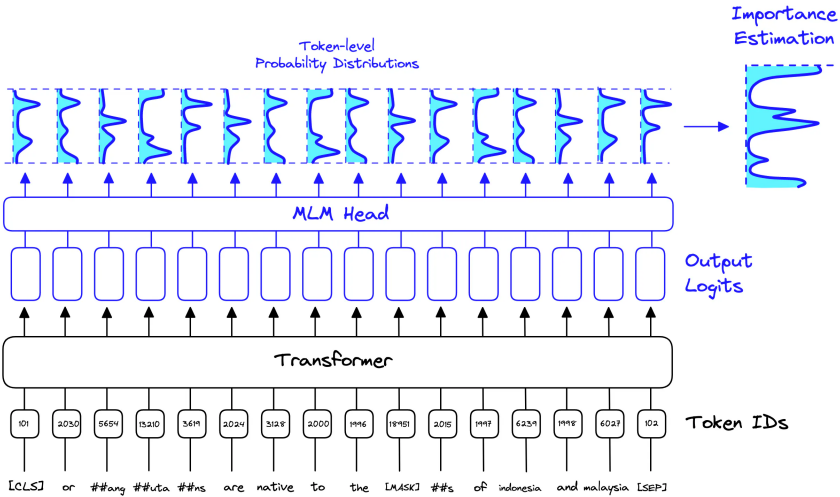
Masked Language Modeling Head



SPLADE: Importance Estimation

- The MLM head gives us a probability distribution for each token, whether or not they have been masked. These distributions are aggregated to give the importance estimation.
- SPLADE takes all these distributions and aggregates them into a single distribution called the importance estimation w_j.
- This importance estimation is the sparse vector produced by SPLADE. We can combine all these probability distributions into a single distribution that tells us the relevance of every token in the vocab to our input sentence.
- $w_j = \sum_{i \in t} log(1 + ReLU(w_{ij}))$
- Where:
  - $i \in t$: Every token i in the input set of tokens t.
  - w_ij: Every predicted weight for all tokens j in the vocab V, for each token i.

SPLADE: Importance Estimation



SPLADE: Importance Estimation

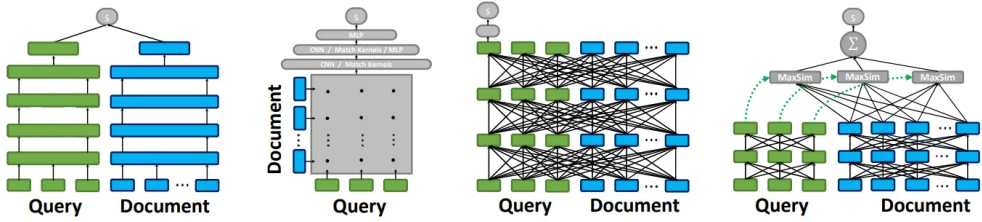
- Importance estimation allows us to identify relevant tokens that do not exist within the input sentence.
- For example, if we mask the word rainforest, we may return high predictions for the words jungle, land, and forest. These words and their associated probabilities would then be represented in the SPLADE-built sparse vector.
- To compare the vectors, we can use cosine or dot-product similarity.
- SPLADE are considered to be hybrid between sparse and dense representations

Dense Representations for Ranking

- By using dense models, we can search based on semantic meaning rather than term matching.
- We need vast amounts of data to fine-tune dense embedding models; without this, they lack the performance of sparse methods. This is problematic for niche domains where data is hard to find and domain-specific terminology is important.
- Dense retrieval techniques need to address two challenges:
  - The representation problem, or the design of the encoders η, to accurately capture the 'meaning' of queries and texts from the corpus for the purposes of ranking; and,
  - The comparison problem, or the design of ϕ, which involves a balance between what can be efficiently computed at scale and what is necessary to capture relevance in terms of the dense representations.
- The dense retrieval approaches are typically architecturally similar to representation-based approaches.

Representation vs. Interaction-based Ranking

- Representation based: query and documents are first translated to embedding space individually without knowledge of each other and then matched, risk of losing exact match signals
- Interaction based: model interaction between query words and document terms explicitly, exact match signals are not lost.

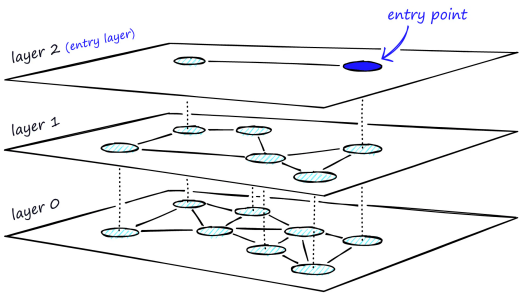


Nearest Neighbor Search

- Ranking in the representation-based architectures is often done using nearest neigbor search
- We assume the existence of a corpus of texts $C = d_i$, it is possible to precompute the output of $\eta_d(\cdot)$ for all d
- $\eta_q(q)$, must be computed at query time
- The ranking problem (i.e. the nearest neghbor search) is to find the top k most similar η_i vectors measured in terms of ϕ.
- The simplest solution to the nearest neighbor search problem is to scan all the η_i vectors and brute force compute $\phi(\eta_q, \eta_i)$ – it is too slow for larger collections

Aproximate Nearest Neighbor Search

- Scalable solutions to the nearest neighbor search problem are based on approximations, hence approximate nearest neighbor (ANN) search
- Methods based on hierarchical navigable small world (HNSW) graphs represent the current state of the art in ANN search
  - A popular open-source library for ANN search is Faiss by Facebook



HNSW

- Vector search using Navigable Small World (NSW) graphs
- The idea is that if we take a proximity graph but build it so that we have both long-range and short-range links
- Then, search times are reduced to logarithmic complexity.

HNSW cont.

- When searching an NSW graph, we begin at a pre-defined entry-point. This entry point connects to several nearby vertices. We identify which of these vertices is the closest to our query vector and move there.
- We repeat the greedy-routing search process of moving from vertex to vertex by identifying the nearest neighboring vertices in each friend list. Eventually, we will find no nearer vertices than our current vertex — this is a local minimum and acts as our stopping condition.
- The routing consists of two phases. We start with the "zoom-out" phase where we pass through low-degree vertices (degree is the number of links a vertex has) — and the later "zoom-in" phase where we pass through higher-degree vertices.

Vector databases

- A Vector Database is a specialized type of database designed to efficiently store and manage high-dimensional data represented as vectors.
- They excel in similarity searches
- Might contain texts, songs, images, ...
- They are based on the ANN search
- E.g. Pinecone, Weaviate, Elasticsearch, ...

Negative sampling

- For retrieval problems, usually, the positive examples are explicitly available while negative examples are to be sampled from a large corpus based on some strategy. Negative sampling is a key challenge during representation learning of a dense retriever.
- There exists different sampling strategies:
  - Explicit Negatives: e.g. documents that were displayed but not clicked, as negatives.
  - Random Negatives: randomly sample documents from the corpus for each query. The random documents can be completely irrelevant to the query and often do not produce hard enough negative examples.
  - BM25 Negatives: take the top documents returned by BM25 which do not contain the answer, but still match the input query tokens.
  - Approximate Nearest Neighbors: after every few thousand steps of training, use the current model to re-encode and re-index the documents in the corpus. Then, retrieve the top-k documents for the training queries and used them as negatives for the following training iterations.
  - In-batch Negatives: select relevant documents of other queries in the same batch. Can be memory-efficient to use.

Bi-encoders for Dense Retrieval: DPR and ANCE

- Dense passage retriever (DPR) and the approximate nearest neighbor negative contrastive estimation (ANCE) are ones of the earliest variants of dense retrieval
- DPR uses "retriever–reader" architecture for question answering, which uses a passage retriever that selects candidate texts from a corpus, which are then passed to a reader to identify the exact answer spans.
- Retriever adopts a bi-encoder design for dense retrieval: it uses separate encoders for the query and texts from the corpus, the comparison function ϕ, is defined in terms of inner product between CLS vectors of query and document
- DPR uses random, BM25, and in-batch negatives, ANCE uses Approximate Nearest Neighbor negatives
- The bi-encoders for dense retrieval based on simple inner-product comparisons are not as effective as cross-encoders, they are generally more effective than sparse retrieval

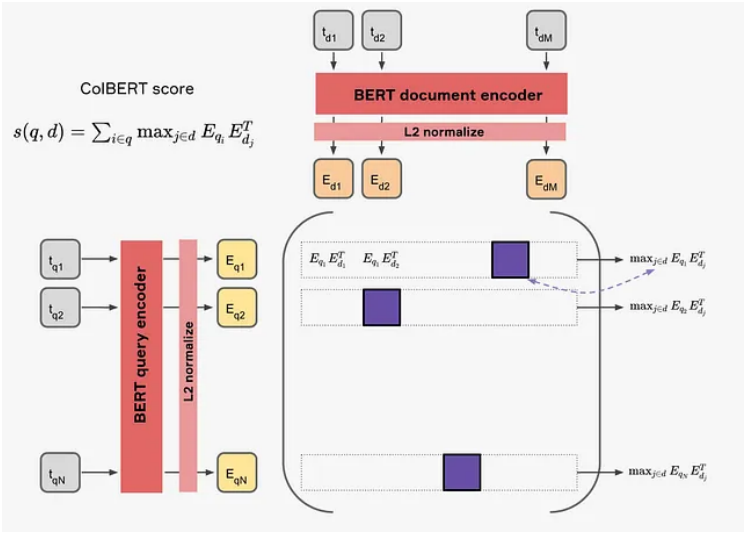
Per-Token Representations and Late Interactions: ColBERT

- Contextualized Late Interaction over BERT
- Late interaction allows efficient and precise retrieval by processing queries and documents separately until the final stages of the retrieval process.
- Cross-encoders process pairs of queries and documents together, making them highly accurate but less efficient for large-scale tasks due to the computational cost of evaluating every possible pair.
- ColBERT aims to improve the efficiency, while keeping some interactive information
- The key operations include a batch dot-product to compute term-wise similarities, max-pooling across document terms to find the highest similarity per query term, and summation across query terms to derive the total document score, followed by sorting the documents based on these scores.

Per-Token Representations and Late Interactions: ColBERT

- The function works by first computing the similarity between each individual query token and each document token using a similarity metric like cosine similarity on their contextualized embeddings.
- The maximum of these token-level similarities are then considered as the final relevance score between the query and document.
- This maximum similarity approach allows ColBERT to identify relevant documents that may not have an exact match with the query, as long as they contain at least one highly relevant term or phrase.
- Even if most tokens don't match well, the presence of a single token with high similarity can boost the overall relevance score. This makes ColBERT effective for retrieval scenarios where queries and relevant documents use different wording or expressions.

ColBERT Architecture



Multidimensional Comparison

	Hardware				Performance		
	GPU	CPU	RAM	Instance	Latency	Cost	Success@10
BM25	0	1	4	m6gd.med	11	$0.14	38.6
BM25	0	1	32	x2gd.lrg	10	$0.48	38.6
DPR					146	$6.78	52.1
ColBERTv2-S					206	$9.58	68.8
ColBERTv2-M					321	$14.90	69.6
ColBERTv2-L					459	$21.30	69.7
BT-SPLADE-L					46	$2.15	66.3
BM25	1	16	32	p3.8xl	9	$29.94	38.6
DPR					18	$61.06	52.1
ColBERTv2-S					27	$90.41	68.8
ColBERTv2-M					36	$123.35	69.6
ColBERTv2-L					55	$187.24	69.7
BT-SPLADE-L					33	$112.87	66.3

Retrieval-Augmented Generation

Problems with LLMs

- Knowledge cutoffs: parameters are usually only updated to a particular time
- Private data: data stored in private text or data repositories are not suitable for training
- Learning failures: even for data that the model was trained on, it might not be sufficient to get the right answer
- Verifiability issues: It is hard to tell if the answer is correct
- Hallucinations

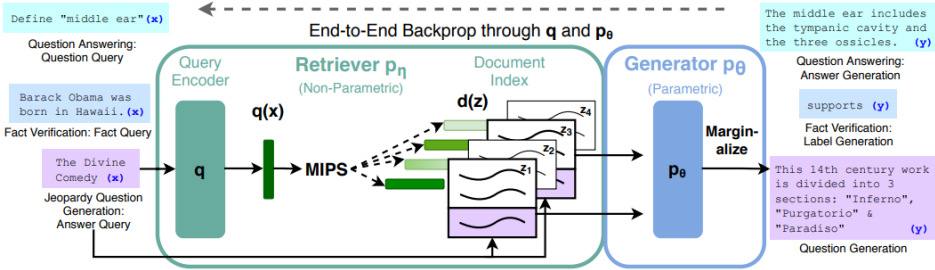
RAG

- Retrieval augmented generation (RAG) is a technique that combines information retrieval with language model generation to improve the accuracy and relevance of the generated text, and to better ground the model's response in evidence.
- In RAG, a language model is augmented with an external knowledge base or a set of documents that is passed into the context window.

RAG Components

- Retrieval Component: The retrieval component is responsible for searching and selecting relevant information from a database or corpus of documents. This component utilizes techniques like document indexing, query expansion, and ranking to identify the most suitable documents based on the user's query.
- Generation Component: Once the relevant documents are retrieved, the generation component takes over. It leverages LLMs to process the retrieved information and generate coherent and contextually accurate responses. This component is responsible for converting retrieved facts into human-readable answers.
- Retrieval and generation should be done end2end
- Knowledge base with the documents to retrieve from

RAG Architecture[1]



[1] Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

Summary

- Multistage architectures
- Query and document expansions using embeddings
- Dense representations, DPR, Splade, Colbert
- RAG

Reading and References

- *Pretrained Transformers for Text Ranking: BERT and Beyond*, Jimmy Lin, Rodrigo Nogueira, and Andrew Yates[2]

[2] https://arxiv.org/pdf/2010.06467