Neural Information Retrieval

[DAT640] Information Retrieval and Text Mining

Ivica Kostric & Weronika Lajewska
University of Stavanger

10.10.2022

In this module

1. Interaction-focused ranking system
2. Learned sparse retrieval
3. Fusion strategies

Interaction-focused ranking system

Interaction-focused ranking system

- Main characteristics and high-level architecture
- Convolutional Neural Networks
- Transformer-based models

Interaction-focused Ranking System

- Interaction-focused ranking systems successfully improve the effectiveness of IR systems in the ad-hoc ranking task
- They use the word and n-gram relationships across a query and a document
- Both a query q and a document d are received as an input, and an output is a query-document representation $\mu(q, d)$

Re-ranking pipeline architecture

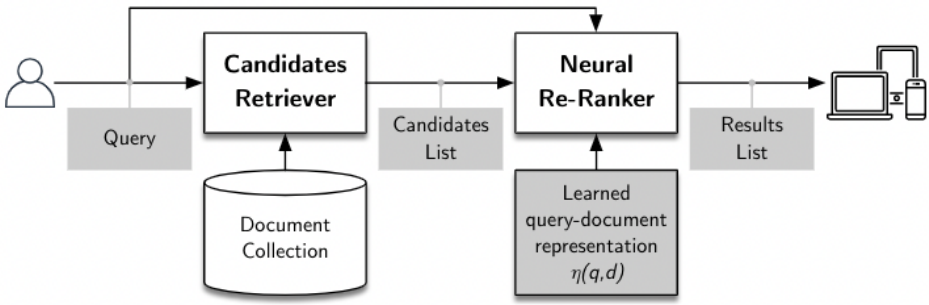


Figure: Re-ranking pipeline architecture for interaction-focused neural IR systems[1]

[1] https://arxiv.org/pdf/2207.13443.pdf

Retrieval architecture

Question

Why can't we just use them directly on the document collection to rank all documents matching a query?

Retrieval architecture - cont.

- Computationally very expensive, because the interaction has to be computed ad-hoc
- We deploy interaction-focused systems in the pipeline architecture only on the first top N candidates:
  - We conduct a preliminary ranking stage to retrieve a limited number of candidates (usually N=1000)
  - We re-rank the candidates using a more expensive neural re-ranking system

Interaction-focused Ranking System

- Generally done using deep neural networks
- Two neural network architectures have been investigated to build a representation of these relationships:
  - convolutional neural networks (CNNs)
  - transformers

Interaction-focused ranking system

- Main characteristics and high-level architecture
- Convolutional Neural Networks
- Transformer-based models

Convolutional Neural Networks (CNNs)

- A convolutional neural network is a family of neural networks designed to capture local patterns in structured inputs, such as images and texts
- The core component of a convolutional neural network is the convolution layer, used in conjunction with feed forward and pooling layers
- A convolutional layer can be seen as a small linear filter, sliding over the input and looking for *proximity patterns* (such as unigrams, bigrams, and so on)

CNN: architecture

- Several neural models employ convolutional neural networks over the interactions between queries and documents to produce relevance scores
- Typically, in these models, the word embeddings of the query and document tokens are aggregated into an *interaction matrix*
- On top of *interaction matrix* the convolutional neural networks are used to learn hierarchical *proximity patterns*
- The final top-level *proximity patterns* are fed into a feed forward neural network to produce the relevance score $s(q,d)$ between the query q and the document d

Interaction-focused model based on CNN

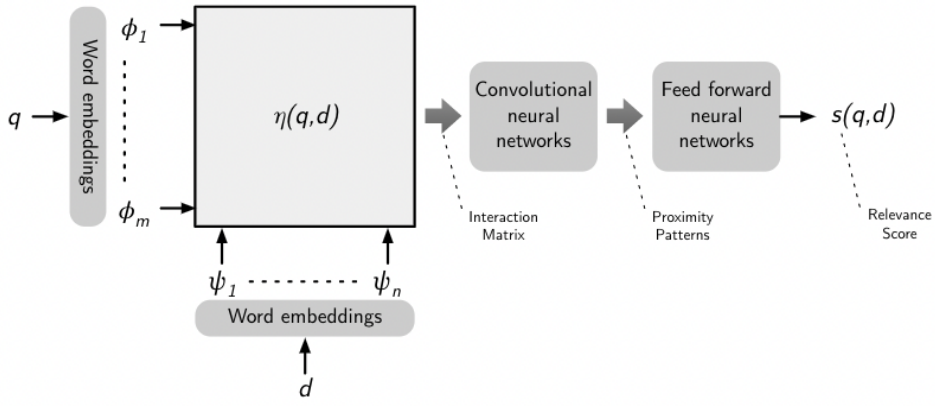


Figure: Scheme of an interaction-focused model based on convolutional neural networks.[2]

[2] https://arxiv.org/pdf/2207.13443.pdf

CNN: interaction matrix

- To construct the *interaction matrix* we need to:
  - tokenize the query q and the document d into m and n tokens respectively
  - map each token to a corresponding static word embedding
  - compute cosine similaritites between a query token embedding and a document token embedding
- Each entry in the *interaction matrix* is the cosine similarity corresponding to the given query and document

Interaction-focused ranking system

- Main characteristics and high-level architecture
- Convolutional Neural Networks
- Transformer-based models

Recap on static and contextualized word embeddings

- Static word embeddings map words with multiple senses into an average or most common-sense representation based on the training data used to compute the vectors
- The static word embedding of a word does not change with the other words used in a sentence around it
- In contextualized word embeddings, the representation of each token is conditioned on the context in which it is considered.

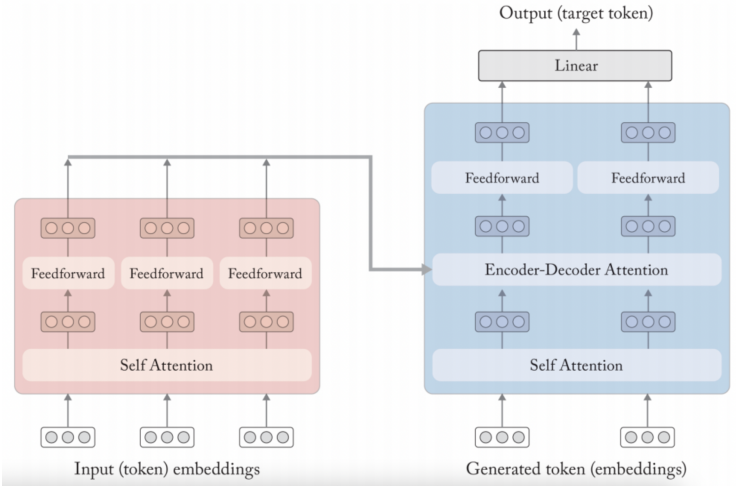
Transformers

- The transformer is a neural network designed to explicitly take into account the context of arbitrary long sequences of text
- Transformers are an example of *sequence-to-sequence* models that transforms an input vectors $(x_1, ..., x_n)$ to some output vectors $(y_1, ..., y_n)$ of the same length
- Transformers are made up of stacks of transformer blocks, which are multilayer networks made by combining simple linear layers, feedforward networks, and **attention layers**
- Attention is a key innovation of the transformers which allows to directly extract and use information from arbitrarily long contexts

Transformers

- Originally, the *sequence-to-sequence* neural network is composed of two parts: an *encoder* model, which receives an input sequence and builds a contextualised representation of each input element, and a *decoder* model, which uses these contextualised representations to generate a task-specific output sequence
- Three variants made for different purposes
  - Encoder-decoder architecture (e.g., machine translation)
  - Encoder only architecture (e.g., generating contextualized embeddings)
  - Decoder only architecture (e.g., language models)

High-level architecture of Transformer model [3]



[3] https://www.morganclaypool.com/doi/abs/10.2200/S01057ED1V01Y202009HLT047

Transformers: self-attention

At the core of an attention-based approach is the ability to compare an item of interest to a collection of other items in a way that reveals their relevance in the current context. A self-attention layer allows the network to take into account the relationships among different elements in the same input.

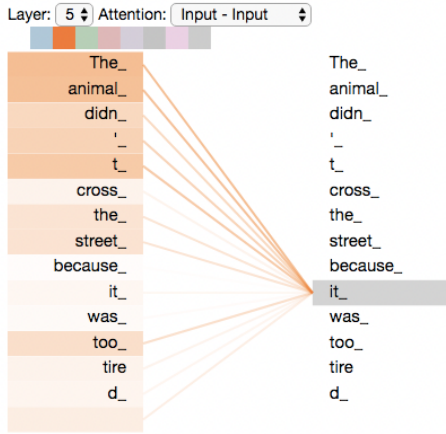


Figure: Self-attention at high level [a]

[a] https://jalammar.github.io/

Transformers: self-attention - cont.

- A self-attention layer maps input sequences $(x_1, ..., x_n)$ to output sequences of the same length $(y_1, ..., y_n)$
- Self attention differs in encoder and decoder modules
- The transformers in the *encoder* employ bidirectional self-attention layers on the input sequence or the output sequence of the previous transformer
- The transformers in the decoder employ causal self-attention on the previous decoder transformer's output and bidirectional cross-attention of the output of the final encoder transformer's output
  - In encoder module, the model has access to all of the inputs
  - In decoder module, the model has access to all of the inputs up to and including the one under consideration, but no access to information about inputs beyond the current one
- In addition, the computation performed for each item is independent of all the other computations $\rightarrow$we can easily parallelize both forward inference and training of such models

Transformers: self-attention - cont.

- The simplest form of comparison between elements in a self-attention layer is a dot product
- Let's refer to the result of this comparison as a score: $score(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$
- We can compute these scores for all pairs i, j (or $j \leq i$ in decoder self-attention) and normalize them with a *softmax* to create a vector of weights α_{ij}, that indicates the proportional relevance of each input to the input element i that is the current focus of attention:

$$\alpha_{ij} = \forall_j softmax(score(\mathbf{x}_i, \mathbf{x}_j)) = \forall_j \frac{exp(score(\mathbf{x}_i, \mathbf{x}_j))}{\sum_k exp(score(\mathbf{x}_i, \mathbf{x}_k))}$$

- Given the proportional scores in α, we then generate an output value y_i by taking the sum of the inputs seen so far, weighted by their respective α value $\mathbf{y}_i = \sum_j \alpha_{ij} \cdot \mathbf{x}_j$

Transformers: self-attention - cont.

- Transformers use a bit more sophisticated way of representing how words can contribute to the representation of longer input
- Input embedding can play three different roles during the course of the attention process:
  - As the current focus of attention when being compared to all of the other query preceding inputs →**query**
  - As a preceding input being compared to the current focus of attention →**key**
  - As a value used to compute the output for the current focus of attention →**value**
- To capture these three different roles, transformers introduce weight matrices $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$. These weights will be used to project each input vector $\mathbf{x}_i$ into a representation of its role as a key, query, or value

$$\mathbf{q}_i = \mathbf{W}^Q \cdot \mathbf{x}_i; \mathbf{k}_i = \mathbf{W}^K \cdot \mathbf{x}_i; \mathbf{v}_i = \mathbf{W}^V \cdot \mathbf{x}_i$$

Transformers: self-attention - cont.[4]

Given these projections, the score between the current focus of attention, $\mathbf{x}_i$ and an element in the context, $\mathbf{x}_j$ consists of a dot product between its query vector $\mathbf{q}_i$ and the context element's key vectors $\mathbf{k}_j$:
$score(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j$
The ensuing softmax calculation resulting in α_{ij} remains the same, but the output calculation for $\mathbf{y}_i$ is now based on a weighted sum over the value vectors v:
$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ji} \cdot \mathbf{v}_j$

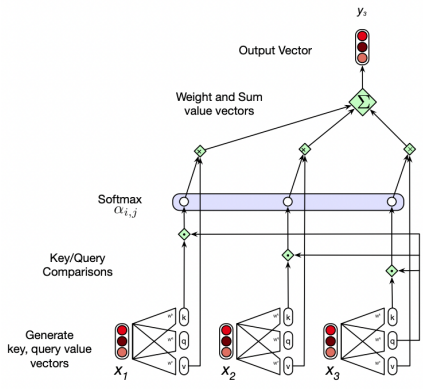


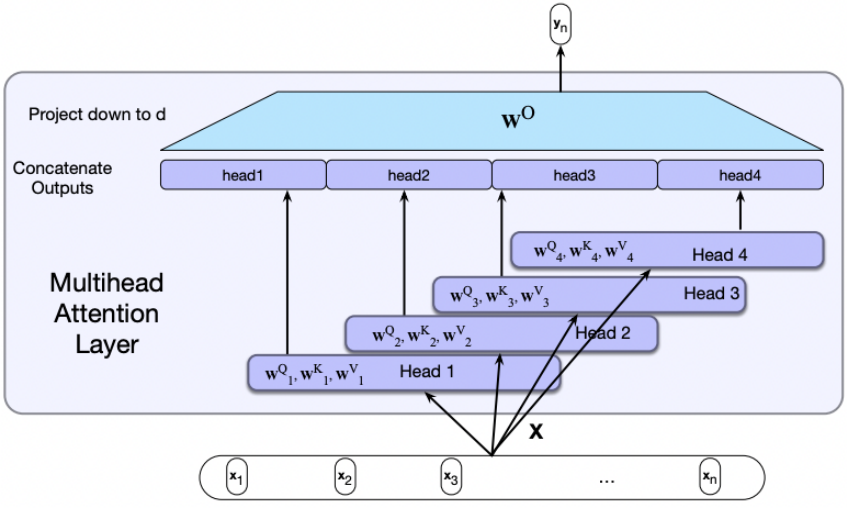
Figure: Self-attention when considering the input $\mathbf{x}_3$

[4] https://arxiv.org/pdf/2207.13443.pdf

Transformers: multihead self-attention

- The different words in a sentence can relate to each other in many different ways simultaneously
- It would be difficult for a single transformer block to learn to capture all of the different kinds of parallel relations among its inputs
- Transformers address this issue with *multihead self-attention layers*
- *Multihead self-attention layers* are sets of self-attention layers, called heads, that reside in parallel layers at the same depth in a model, each with its own set of parameters
- Each of the multihead self-attention layers is provided with its own set of key, query and value weight matrices
- The outputs from each of the layers are concatenated and then projected down to the same size as the input so layers can be stacked

Transformers: multihead self-attention[5]

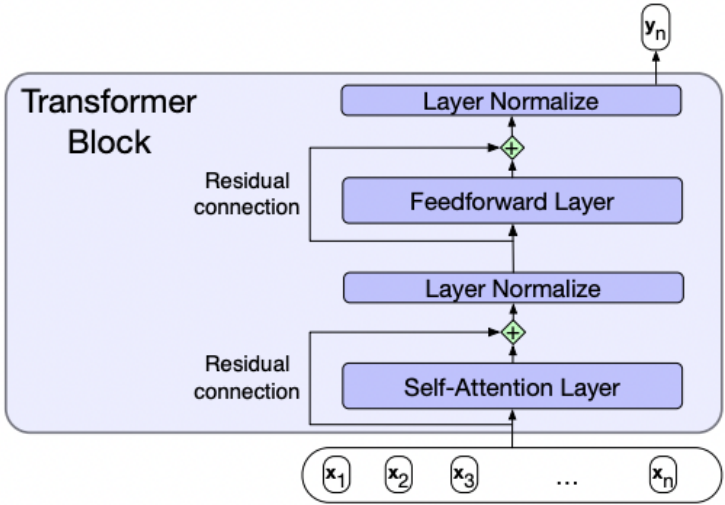


[5] https://web.stanford.edu/~jurafsky/slp3/

Transformers: transformer block

- The self-attention calculation lies at the core of what's called a *transformer block*, which, in addition to the self-attention layer, includes additional feedforward layers, residual connections, and normalizing layers
- A standard *transformer block* consists of a single attention layer followed by a fully-connected feedforward layer with residual connections and layer normalizations following each

Transformers: transformer block[6]



[6] https://web.stanford.edu/~jurafsky/slp3

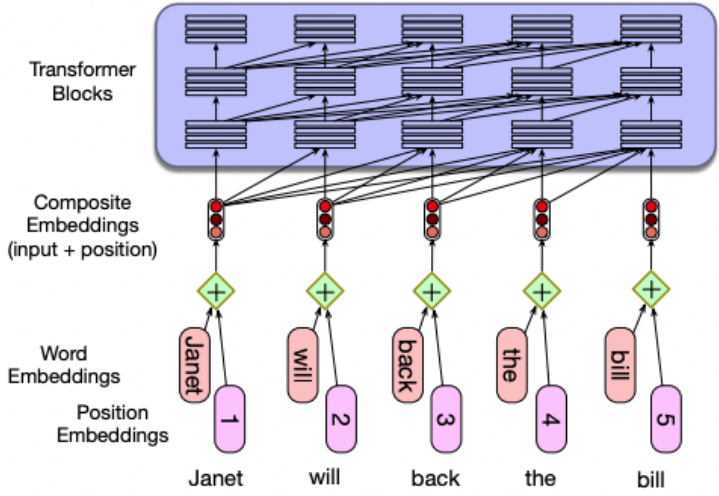
Transformers: transformer block - cont.

- In deep networks, residual connections are connections that pass information from a lower layer to a higher layer without going through the intermediate layer
- Allowing information from the activation going forward and the gradient going backwards to skip a layer improves learning and gives higher level layers direct access to information from lower layers
- Residual connections in transformers are implemented by adding a layer's input vector to its output vector before passing it forward
- *Layer normalization* is used to improve training performance in deep neural networks by keeping the values of a hidden layer in a range that facilitates gradient-based training

Transformers: positional embeddings

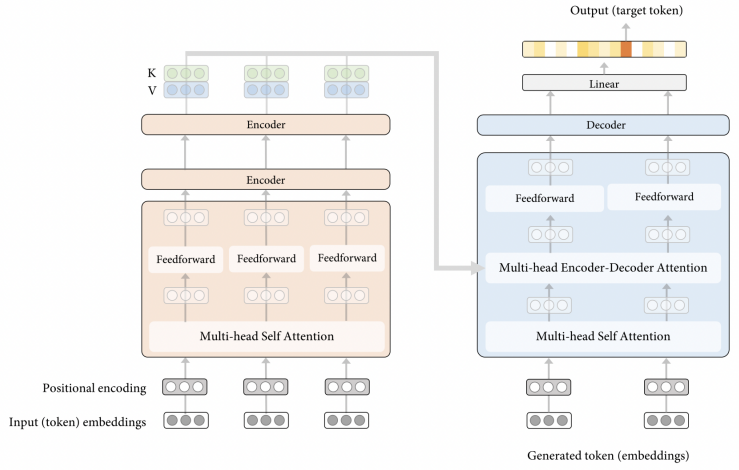
- Transformers modify the input embeddings by combining them with *positional embeddings* specific to each position in an input sequence
- The simplest way to model the possition is to:
  - start with randomly initialized embeddings corresponding to each possible input position up to some maximum length
  - and learn the positional embeddings along with other parameters during training
- To produce an input embedding that captures positional information, we just add the word embedding for each input to its corresponding positional embedding

Transformers: positional embeddings[7]



[7] https://web.stanford.edu/~jurafsky/slp3

High-level architecture of Transformer-based encoder-decoder model[8]



[8] https://aclanthology.org/2020.coling-tutorials.2/

Pre-trained Language Models

- Transformer-based models are usually trained using massive text data to obtain *pre-trained language models*
- It allows the model to learn general-purpose knowledge about a language that can be adapted afterwards to a more specific *downstream task*
- This approach is called *transfer learning*
- In *transfer learning*, a pre-trained language model is used as initial model to *fine-tune* it on a domain-specific, smaller training dataset for the downstream target task
- *Fine-tuning* is the procedure to update the parameters of a pre-trained language model for the domain data and target task

Transfer learning[9]

[9] https://arxiv.org/pdf/2207.13443.pdf

Demo

HuggingFace Transformers [10]

[10] https://transformer.huggingface.co

Fine-tuning interaction-focused systems

- The pre-trained language models adopted in IR require a fine-tuning on a specific downstream task
- Given an input query-document pair *(q,d)* a neural IR model $M(\theta)$, parametrised by θ, computes $s_\theta(q, d) \in \mathbb{R}$, the score of the document d w.r.t. the query q
- We want to predict $y \in \{+, -\}$ from $(q, d) \in Q \times D$, where "$-$" stands for non-relevant and "$+$" for relevant
- We perform the classification by assuming a joint distribution p over $\{+, -\} \times Q \times D$ from which we can sample correct pairs $(+, q, d) \equiv (q, d^+)$ and $(-, q, d) \equiv (q, d^-)$
- Training objective: *the score function must assign a high score to a relevant document and a low score to a non-relevant document*

Fine-tuning interaction-focused systems - cont.

- We need to find θ^* that minimises the (binary) cross entropy l_{CE} between the conditional probability $p(y|q,d)$ and the model probability $p_\theta(y|q,d)$:

$$\theta^* = \underset{\theta}{\text{argmin}}\, \mathbb{E}[l_{CE}(y,q,d)]$$

where the expectation is computed over $(y,q,d) \sim p$, and the cross entropy is computed as:

$$l_{CE}(y,(q,d)) = \begin{cases} -log(s_\theta(q,d)) & \text{if y=+} \\ -log(1-s_\theta(q,d)) & \text{if y=-} \end{cases} \quad (1)$$

Fine-tuning interaction-focused systems - cont.

- Typically, a dataset T available for fine-tuning pre-trained language models for relevance scoring is composed of a list of triples (q, d^+, d^-), where q is a query, d^+ is a relevant document for the query, and d^- is a non-relevant document for the query
- In this case, the expected cross entropy is approximated by the sum of the cross entropies computed for each triple:

$$\mathbb{E}[l_{CE}(y, (q, d))] \approx \frac{1}{2|T|} \sum_{(q,d^+,d)\in T} (-log(s_\theta(q, d^+)) - log(1 - s_\theta(q, d^-)))$$

Transformers: sequence-to-sequence models

- In neural IR, two specific instances of the sequence-to-sequence models have been studied:
  - encoder-only models
  - encoder-decoder models

Transformers: encoder-only models

- *Encoder-only* models receive as input all the tokens of a given input sentence, and they compute an output contextualised word embedding for each token in the input sentence
- Representatives of this family of models include BERT, RoBERTa, and DistilBERT
- Encoder-only models are trained as *masked language models (MLM)*
- MLM training focuses on learning to predict missing tokens in a sequence given the surrounding tokens

BERT model

- BERT input text is tokenised using the WordPiece sub-word tokeniser
- The vocabulary V of this tokeniser is composed of 30,522 terms, where the uncommon/rare words, e.g.,*goldfish*, are splitted up in sub-words, e.g., *gold##* and *##fish*
- The first input token of BERT is always the special *[CLS]* token, that stands for "classification"
- BERT accepts as input other special tokens, such as *[SEP]*, that denotes the end of a text provided as input or to separate two different texts provided as a single input
- BERT accepts as input at most 512 tokens, and produces an output embedding in $\mathbb{R}^l$ for each input token
- The most commonly adopted BERT version is *BERT-base*, which stacks 12 transformer layers, and whose output representation space has $l = 768$ dimensions

BERT model - cont.

- Given a query-document pair, both texts are tokenised into token sequences $q_1, ..., q_m$ and $d_1, ..., d_n$
- Then, the tokens are concatenated with BERT special tokens to form the following input configuration that will be used as BERT input:

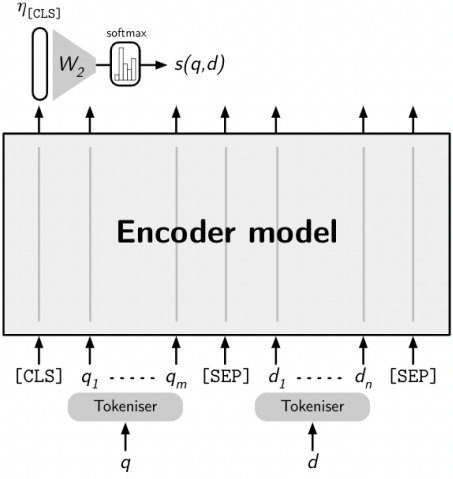
$$[CLS]q_1, ..., q_m[SEP]d_1, ..., d_n[SEP]$$

- The self-attention layers in the BERT encoders are able to take into account the semantic interactions among the query tokens and the document tokens
  - The attention mechanism in encoder models is bi-directional
- The output embedding $\mu_{[CLS]} \in \mathbb{R}^l$, corresponding to the input *[CLS]* token, serves as a contextual representation of the query-document pair as a whole

Fine-tuning BERT model for computing relevance score

- The query and the document are processed by BERT to generate the output embedding $\mu_{[CLS]} \in \mathbb{R}^l$
- $\mu_{[CLS]}$ is multiplied by a learned set of classification weights $W_2 \in \mathbb{R}^{2xl}$ to produce two real scores z_0 and z_1
- The scores z_0 and z_1 are transformed into a probability distribution p_0 and p_1 over the non-relevant and relevant classes
- The probability corresponding to the relevant class, conventionally assigned to label 1, i.e.,p_1, is the final relevance score

BERT classification model for ad-hoc ranking[11]



[11] https://arxiv.org/pdf/2207.13443.pdf

Exercise

E10-1 - WordPiece Tokenizer

Transformers: encoder-decoder models

- In the *encoder-decoder* model the encoder model receives as input all the tokens of a given sequence and builds a contextualised representation, and the decoder model sequentially accesses these embeddings to generate new output tokens, one token at a time
- Representatives of this family of models include BART and T5
- Encoder-decoder models are trained as *casual language models (CLM)*
- CLM training focuses on predicting the next token in an output sequence given the preceding tokens in the input sequence
- Encoder-decoder models use *prompt learning* which converts the relevance score computation task into a cloze test, i.e., a fill-in-the-blank problem

Encoder-decoder models - prompt learning

- In *prompt learning* the input texts are reshaped as a natural language template, and the downstream task is reshaped as a cloze-like task
- For example, in topic classification, assuming we need to classify the sentence *text* into two classes c_0 and c_1, the input template can be: $Input : text\ Class : [OUT]$
- Among the vocabulary terms, two label terms w_0 and w_1 are selected to correspond to the classes c_0 and c_1, respectively
- The probability to assign the input text to a class can be transferred into the probability that the input token *[OUT]* is assigned to the corresponding label token:

$$p(c_0|text) = p([OUT] = w_0|Input : text\ Class : [OUT])$$

$$p(c_1|text) = p([OUT] = w_1|Input : text\ Class : [OUT])$$

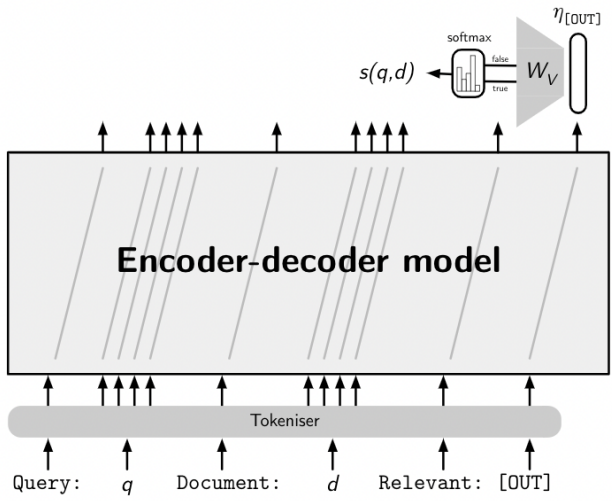
Fine-tuning T5 model for computing relevance score

- In prompt learning approach for relevance ranking using a T5 model, the query and the document texts q and d are concatenated to form the following input template:

$$Query : q \; Document : d \; Relevant : [OUT]$$

- An encoder-decoder model is fine-tuned with a downstream task taking as input this input configuration, and generating an output sequence whose last token is equal to *True* or *False*, depending on whether the document d is relevant or non-relevant to the query q
- The query-document relevance score is computed by normalising only the *False* and *True* output probabilities, computed over the whole vocabulary, with a *softmax* operation

T5 model for ad-hoc ranking[12]

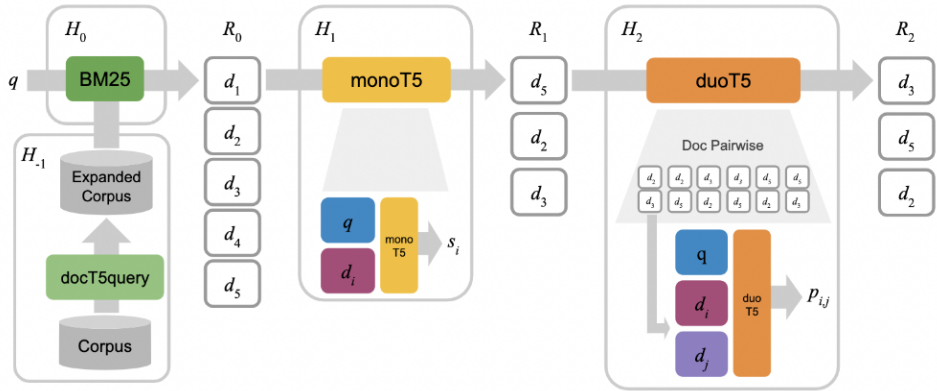


[12] https://arxiv.org/pdf/2207.13443.pdf

Pointwise and pairwise re-ranking

- Re-rankers play a crucial role in pushing the most relevant passages to the top of the ranking
- We distinguish two classes of the reranking methods:
  - *pointwise reranking*, e.g. monoT5, where the relevance of each passage in the ranking with respect to the query is computed independently of other passages $P(Relevant = 1|d_i, q)$
  - *pairwise reranking*, e.g. duoT5, where the relevance is considered in terms of pairs of passages $P(d_i > d_j = 1|d_i, d_j, q)$
- Pointwise reranking with monoT5 is highly effective with different first-stage retrieval results
- Pairwise reranking with duoT5 further improves the effectiveness of monoT5 output, particular in early-precision metrics

Re-ranking with mono/duoT5 [13]



[13] https://www.semanticscholar.org/reader/e08eed9608382beea1febca49119c665fbabd031

Learned sparse retrieval

Learned Sparse Retrieval

- In industry-scale web search, BM25 is a widely adopted baseline due to its trade-off between effectiveness and efficiency
- On the other side, neural IR systems are based on dense representations of queries and documents, that have shown impressive benefits in search effectiveness, but at the cost of query processing times
- In recent years, there have been some proposals to incorporate the effectiveness improvements of neural networks into inverted indexes, with their efficient query processing algorithms, through *learned sparse retrieval* approaches

Learned Sparse Retrieval

- In learned sparse retrieval the transformer architectures are used in different scenarios:
  - document expansion learning
  - impact score learning
  - sparse representation learning

Learned sparse retrieval

- Document Expansion Learning
- Impact Score Learning
- Sparse Representation Learning

Document Expansion Learning

- *Document expansion techniques* address the vocabulary mismatch problem: *queries can use terms semantically similar but lexically different from those used in the relevant documents*
- Traditionally, this problem has been addressed using query expansion techniques, such as relevance feedback and pseudo-relevance feedback
- In *document expansion learning*, sequence-to-sequence models are used to modify the actual content of documents, boosting the statistics of the important terms and generating new terms to be included in a document

Document Expansion Learning: Doc2Query

- Transformer architectures can be used to expand the documents' content to include new terms
- Doc2Query generate new queries for which a specific document will be relevant
- Doc2Query fine-tunes a sequence-to-sequence transformer model by taking as input the relevant document and generating the corresponding query
- Then, the fine-tuned model is used to predict new queries using top k random sampling to enrich the document by appending these queries before indexing

Document Expansion Learning: Doc2Query

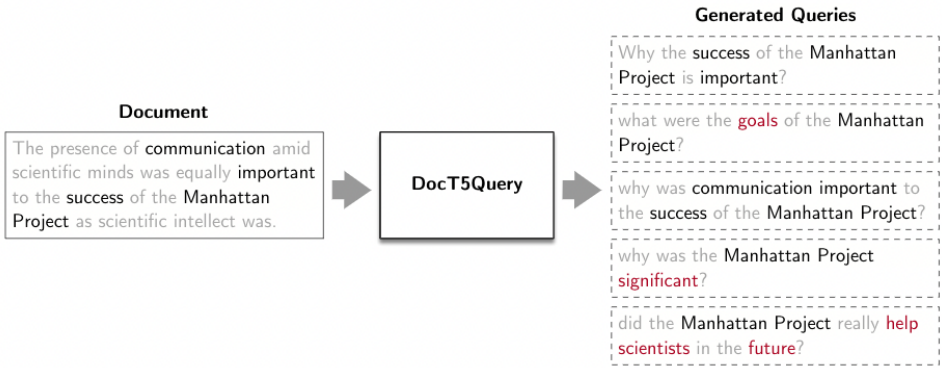


Figure: Example of generated queries using DocT5Query. Black terms denote boosted important terms, red terms denote new important terms not present in the original document[14]

[14] https://arxiv.org/pdf/2207.13443.pdf

Exercise

E10-2 - Document expansion with Doc2Query

Learned sparse retrieval

- Document Expansion Learning
- Impact Score Learning
- Sparse Representation Learning

Impact Score Learning: Recap on inverted index

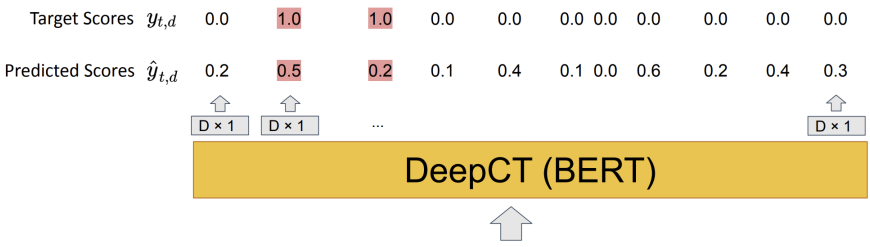
- Classical inverted indexes store statistical information on term occurrences in documents in posting lists, one per term in the collection
- Every posting list stores a posting for each document in which the corresponding term appears in, and the posting contains a document identifier and the in-document term frequency, i.e., a positive integer counting the number of occurrences of the term in the document
- When a new query arrives, the posting lists of the terms in the query are processed to compute the top scoring documents

Impact Score Learning

- The goal of *impact score learning* is to leverage the document embeddings generated by an encoder-only model to compute a single integer value to be stored in postings, and to be used as a proxy of the relevance of the term in the corresponding posting, i.e., its *term importance*
- The simplest way to compute term importance in a document is to project the document embeddings of each term with a neural network into a single-value representation, filtering out negative values with *ReLU* functions and discarding zeros
- The collection vocabulary that is used by impact score learning can be constructed in two different ways: by using the terms produced by the encoder-specific sub-word tokeniser, e.g., by BERT-like tokenisers, or by using the terms produced by a word tokeniser
- These two alternatives have an impact on the final inverted index: in the former case, we have fewer terms, but longer and denser posting lists, while in the latter case, we have more terms, with shorter posting lists and with smaller query processing times

Impact Score Learning: DeepCT[15]

$$\text{loss} = \sum_t (\hat{y}_{t,d} - y_{t,d})^2$$

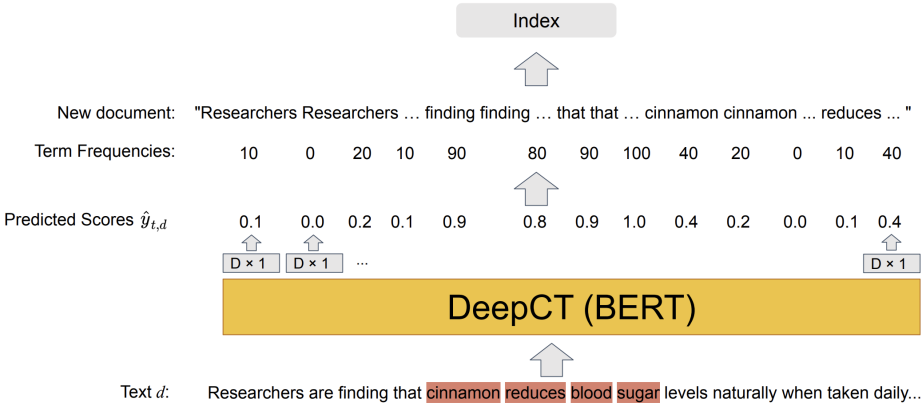


Text d: The Geocentric Theory was proposed by the greeks under the guidance...

Relevant query q: "who proposed the geocentric theory"

[15] http://essir2022.org/slides/andrew-yates.pdf

Impact Score Learning: Indexing with DeepCT[16]



[16] http://essir2022.org/slides/andrew-yates.pdf

Impact Score Learning

- To sum up: in *impact score learning*, the output embeddings of documents provided as input to encoder-only models are further transformed with neural networks to generate a single real value, used to estimate the average relevance contribution of the term in the document

Learned sparse retrieval

- Document Expansion Learning
- Impact Score Learning
- Sparse Representation Learning

Sparse Representation Learning

- Instead of independently learning to expand the documents and then learning the impact score of the terms in the expanded documents, sparse representation learning aims at learning both at the same time
- In general, *sparse representation learning* projects the output embeddings of an encoder-only model into the input vocabulary and computes, for each input term in the document, a language model, i.e., a probability distribution over the whole vocabulary
- These term-based language models capture the semantic correlations between the input term and all other terms in the collection, and they can be used to:
  - expand the input text with highly correlated terms
  - compress the input text by removing terms with low probabilities w.r.t. the other terms

SPLADE: Sparse Lexical and Expansion Model

Key improvement: leverage the **MLM head** for weighting and expansion

➔ Recall that MLM head predicts what term should fill in a [MASK]

Input: a query Q or document D

Output: a sparse vector of dimension |V|, to be indexed

1. For each term in the input, use MLM head to predict scores for|V| terms
2. For each term in the vocabulary V, take maximum score as the term's weight
3. Use weights from #2 to represent the input (Q or D) as a |V| vector; index

[17] `http://essir2022.org/slides/andrew-yates.pdf`

Sparse Representation Learning

- To sum up: in *sparse representation learning*, the output embeddings of documents provided as input to encoder-only models are projected with a learned matrix on the collection vocabulary, in order to estimate the relevant terms in a document
- These relevant terms can be part of the documents or not, hence representing another form of document enrichment

Fusion strategies

Fusion strategies

- Both, sparse retrieval methods and dense retrieval methods have their advantage
- To exploit all of the advantages, we can perform a hybrid retrieval with a sparse method and a dense method
- The final retrieval result will be the “combination” of the two

Fusion strategies: Reciprocal Rank Fusion

- Reciprocal Rank Fusion (RRF) is a simple method for combining the document rankings from multiple IR systems
- Given a set of documents D to be ranked and a set of rankings R, each a permutation on $1,...,|D|$, RRF sorts the documents according to the scoring formula:

$$RRFscore(d \in D) = \sum_{r \in R} \frac{1}{k + r(d)}$$

  - r is the rank of a document d in one ranking
  - k is a smoothing factor that mitigates the impact of high rankings by outlier systems ($k = 60$ in the original paper)
- Intuition: while highly ranked documents are more important, the importance of lower-ranked documents does not vanish (as it would if we use for example an exponential function

Summary

- Interaction-focused ranking systems
  - Main characteristics and high-level architecture
  - Convolutional Neural Networks
  - Transformer-based models
- Learned sparse retrieval
  - Document Expansion Learning
  - Impact Score Learning
  - Sparse Representation Learning
- Fusion strategies

Reading

- *Lecture Notes on Neural Information Retrieval*, Nicola Tonellotto[18]
- *Vector Semantics and Embeddings*, Daniel Jurafsky and James H. Martin [19]

[18] https://arxiv.org/pdf/2207.13443.pdf
[19] https://web.stanford.edu/~jurafsky/slp3/