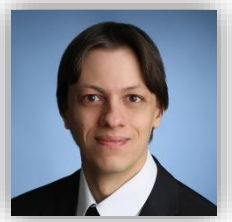
NLP and the Web – WS 2024/2025

Lecture 7
Information Retrieval V

TECHNISCHE UNIVERSITÄT DARMSTADT



Dr. Thomas Arnold
Hovhannes Tamoyan
Kexin Wang

Ubiquitous Knowledge Processing Lab
Technische Universität Darmstadt

Syllabus (tentative)

Nr.	Lecture
01	Introduction / NLP basics
02	Foundations of Text Classification
03	IR – Introduction, Evaluation
04	IR – Word Representation
05	IR – Transformer/BERT
06	IR – Dense Retrieval
07	**IR – Neural Re-Ranking**
08	LLM – Language Modeling Foundations
09	LLM – Neural LLM, Tokenization
10	LLM – Adaption, LoRa, Prompting
11	LLM – Alignment, Instruction Tuning
12	LLM – Long Contexts, RAG
13	LLM – Scaling, Computation Cost
14	Review & Preparation for the Exam

Today

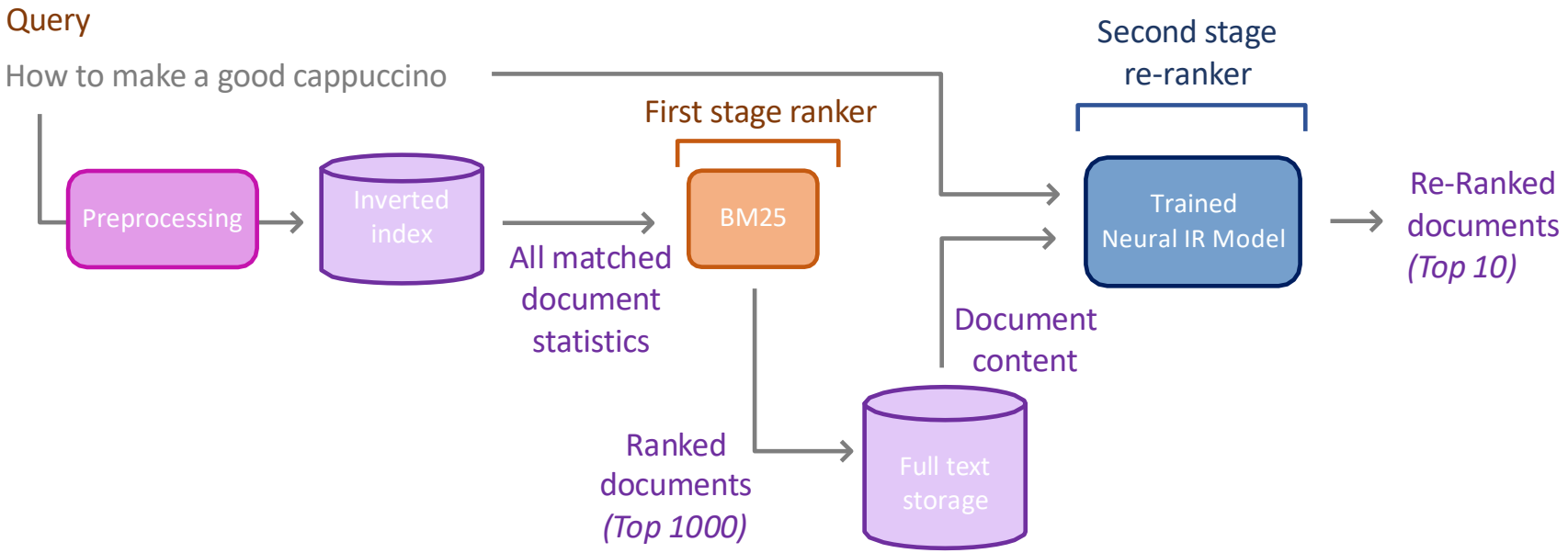
IR – Word Representation / Neural IR

1. **Neural Re-Ranking**
   - Basic Workflow, training & evaluation
   - Basic MatchPyramid approach
2. Re-Ranking with BERT
   - Efficient Models (PreTTR, ColBERT)

Based on:
https://github.com/sebastian-hofstaetter/teaching/tree/master/introduction-to-information-retrieval/lectures

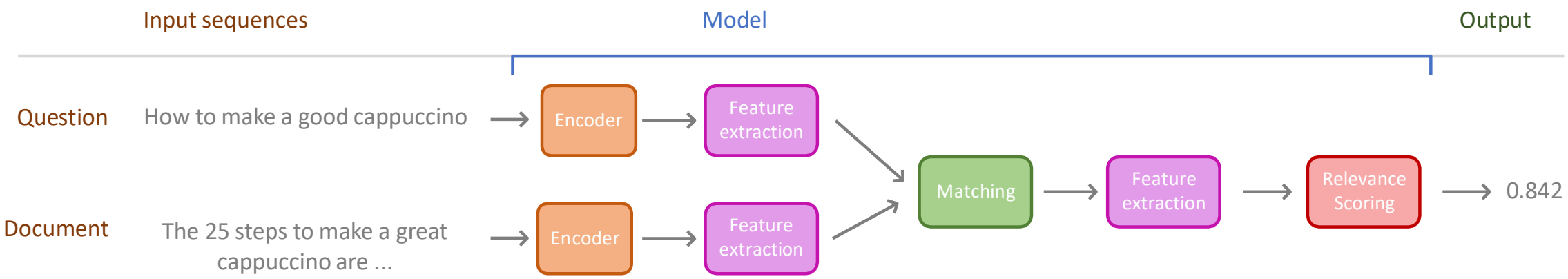
Neural Re-Ranking Models

- Re-rankers: They change the ranking of a pre-selected list of results
  - Same interface as classical ranking methods: $\text{score}(q, d)$
- Query Workflow:



Inside Neural Re-ranking Models

- Core part of re-ranking models is a matching module
  - Operating on a word interaction level



Training (same as Dense Reteival)

- Training is independent from the rest of the search engine operations
  - But it could be done repeatedly to account for temporal shift in the data
- Neural IR models are typically trained with triples (pairwise +,-)
  - Triple: 1 query, 1 relevant, 1 non-relevant document
  - Generate embeddings for query, relevant doc, non-relevant doc
  - Loss function: Maximize margin between rel/non-rel document
- All model components are trained end-to-end
  - Of course we could decide to freeze some parts for more efficient training

Re-Ranking Evaluation

- Scoring per tuple (1 query, 1 document)
- List of tuples is then sorted & evaluated with ranking metric per query (for example: MRR@10)
  - MRR@10 = Mean Reciprocal Rank, stop to look at position 10 or first relevant

- Mismatch: You can’t really compare training loss and IR evaluation metric
  - Training loss is only good for checking at the beginning if your network is not completely broken :) – it should go down very quick and then not change

Today

IR – Word Representation / Neural IR

1 Neural Re-Ranking

- Basic Workflow, training & evaluation
- **Basic MatchPyramid approach**

2 Re-Ranking with BERT

- Efficient Models (PreTTR, ColBERT)

Based on:
https://github.com/sebastian-hofstaetter/teaching/tree/master/introduction-to-information-retrieval/lectures

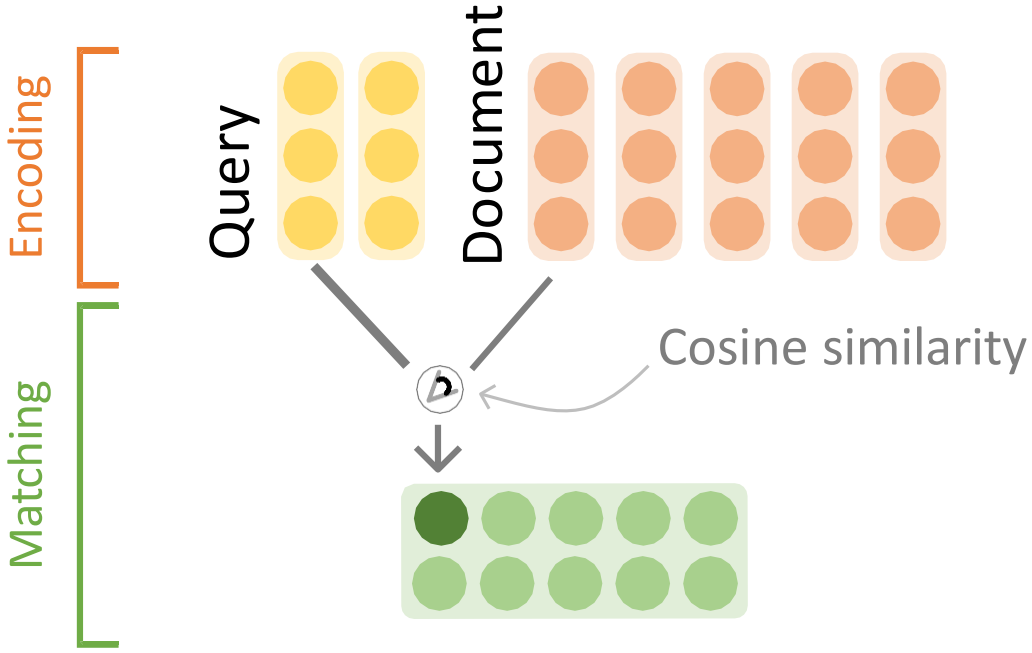
The encoding layer

- Starting point for text processing neural network models
- Word token (id/piece/char based) to dense representation
  - Having word boundaries is important in IR

- <2019: a word embedding
  - Pre-trained Word2vec/Glove
  - Fine-tuned (= trained with the rest of the model)

- 2019+: BERT (huge transformer based model, only pre-trained)
  - Shows very strong results

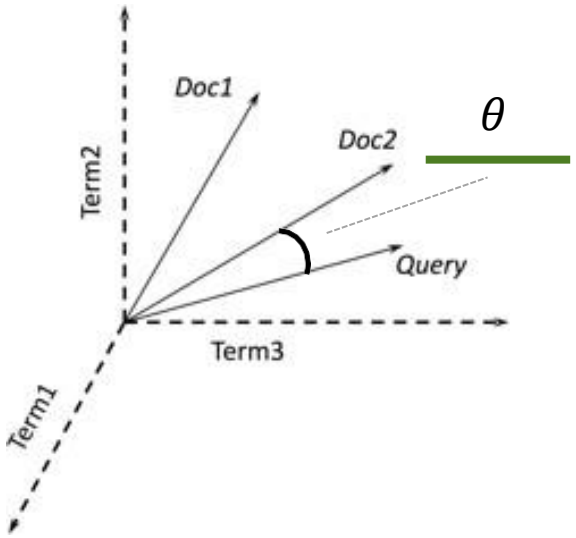
The match matrix



- The core of many early neural IR models*
- Matrix of similarities of individual word combinations
- Only a transformation – not parameterized by itself

* Of course there are other approaches, but we focus on models based on the match matrix

Cosine similarity



$$\text{sim}(d, q) = \cos(\theta)$$
$$= \frac{d \cdot q}{|d||q|}$$

θ	Angle between two vectors
q	Vector of query
d	Vector of document
$d \cdot q$	Dot product

$$= \sum_{1=1,n} d_i * q_i$$

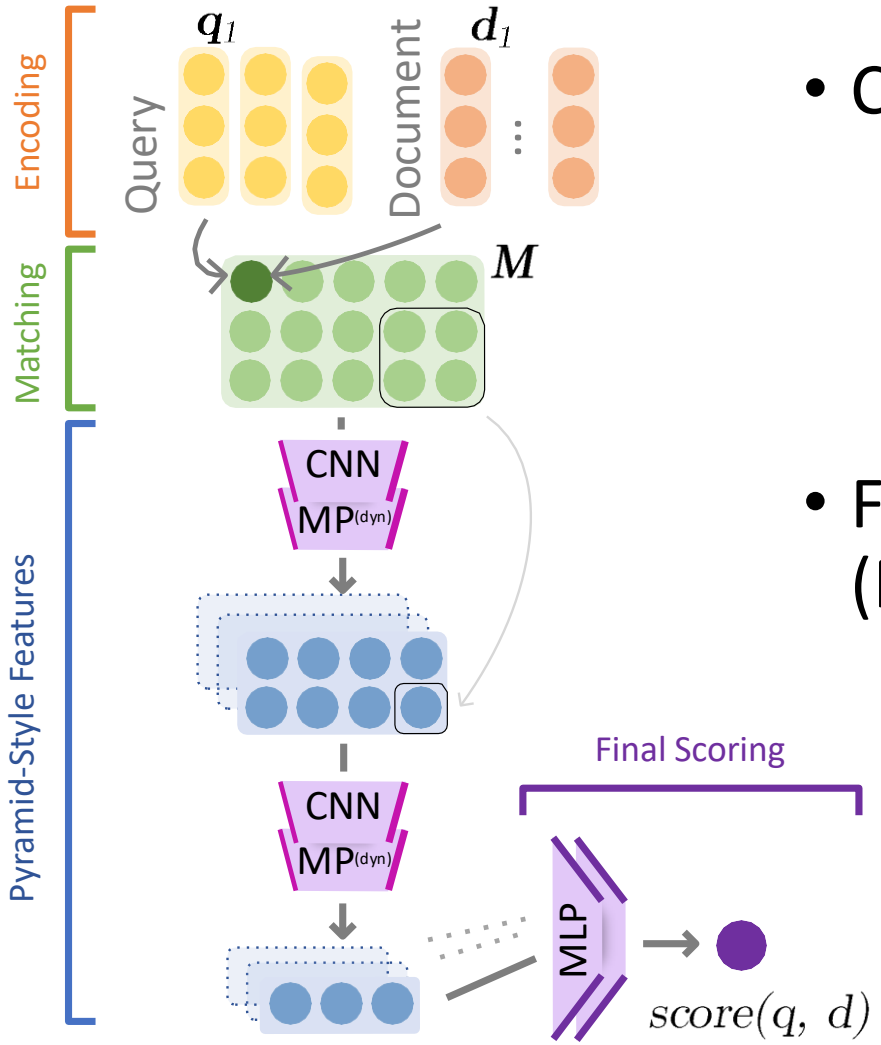
- Cosine similarity measures direction of vectors, but not the magnitude
- Not a distance – but equivalent to Euclidean distance of unit (length=1) vectors

MatchPyramid

- Apply a set of 2D convolutional layers on top of the match matrix
  - Inspired by computer vision: match matrix = image
- Each Conv Layer: 2D-CNN & 2D-Dynamic-Pooling
  - Dynamic Pooling takes care of variable length input to fixed output
- Architecture & effectiveness strongly depends on configuration
  - How many layers
  - Which CNN & pooling kernel sizes
  - Generally: Pooling output becomes gradually smaller (like the Pyramid)

Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2016. Text Matching as Image Recognition. In Proc. of AAAI.

MatchPyramid



- Conv-layers extract local interaction features
  - Max pooling only keeps strong interaction signals (better matches)
  - Different channels can learn different interaction patterns
- Finally a multi-layered feed forward module (MLP) scores the extracted feature vectors

Simplified architecture illustration: Omitted zero-padding (at the start of the pyramid), more layers & channels in practice, last layer can be >1D (nD can be flattened to 1D)

Today

IR – Word Representation / Neural IR

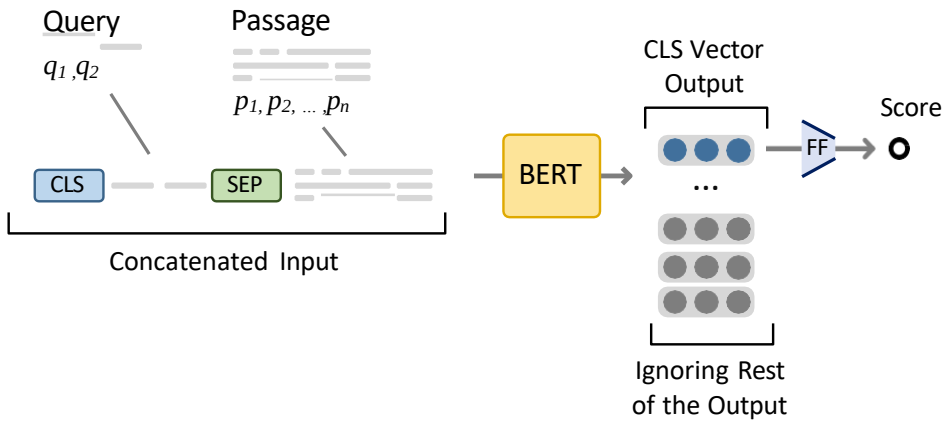
1. Neural Re-Ranking
   - Basic Workflow, training & evaluation
   - Basic MatchPyramid approach
2. **Re-Ranking with BERT**
   - Efficient Models (PreTTR, ColBERT)

Based on:
https://github.com/sebastian-hofstaetter/teaching/tree/master/introduction-to-information-retrieval/lectures

Context of this Lecture

- We are studying ad-hoc re-ranking models
  - Many other applications of Transformers in IR:
    - Document & query expansion
    - QA pipelines
    - Conversational search, using the query history
    - Dense retrieval
    - Knowledge graph-based search
- We are focusing on the efficiency-effectiveness tradeoff
  - How fast or slow is a model compared to the result quality
  - Many other aspects can be studied, f.e.: social biases, domain adaption, or multilingual models (and many more)

BERT Re-Ranking: $BERT_{CAT}$



- Also know as monoBERT, vanilla BERT re-ranking, or simply BERT
- Concatenating the two sequences to fit BERT's workflow
  - [CLS] query [SEP] passage
  - Pool [CLS] token
  - Predict the score with a single linear layer
- Needs to be repeated for every passage

Passage Re-ranking with BERT. Rodrigo Nogueira, Kyunghyun Cho. 2019
https://arxiv.org/abs/1901.04085

BERT_{CAT}

- Simple formula (as long as we abstract BERT):

BERT

$$r = \text{BERT}([CLS]; q_{1..n}; [SEP]; p_{1..m})_{CLS}$$

Concatenation

Scoring

$$s = r * W$$

Starts uninitialized

- We still have the choice of BERT-model

$q_{1..n}$	Query tokens
$p_{1..m}$	Passage tokens
BERT	Pre-trained BERT model
$[\text{CLS}]$ $[\text{SEP}]$	Special tokens
x_{CLS}	Pool the CLS vector
W	Linear Layer (from 768 dims to 1)
s	Output score

The Impact of $BERT_{CAT}$

- This model (first shown by Nogueira and Cho) jumpstarted the current wave of neural IR 🎉
- Works awesome out of the box
  - Concatenating the two sequences to fit BERT's workflow
  - As long as you have time or enough compute it trains easily
- Major jumps in effectiveness across collections and domains
  - But, of course, comes at the cost of performance and virtually no interpretability
  - Larger BERT models roughly translate to slight effectiveness gains at high efficiency cost
    - The problem is we need to repeat the inference by the re-ranking depth!

So how good is $BERT_{CAT}$?

- MSMARCO-Passage
  - MRR@10 from .194 (BM25) to .385 (ALBERT-Large)
  - BERT basically doubles the result quality
- MSMARCO-Document
  - MRR@10 from .252 (BM25) to .384 (DistilBERT with 2K tokens)
- Similar results for TREC-DL ’19, ’20, TripClick
  - Large Training Data Settings

See also the (retired) MSMARCO leaderboard: https://microsoft.github.io/msmarco/
Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation; Hofstätter et al. https://arxiv.org/abs/2010.02666

The Mono-Duo Pattern

- Mono-Duo is a multi-stage process:
  - Mono = score(q, p) – f.e. Top-1000
  - Duo = score(q, p1, p2) – f.e. Top-50 (but needs to be done for all 50^2 pairs)
    - Pairwise comparison: is p1 or p2 more relevant for query q?

- Improves on the single “mono” stage
  - Good for leaderboard settings, showing high performance
- Can use BERT or T5 as base model
  - T5 is also a Transformer based language model, even larger but of course even slower

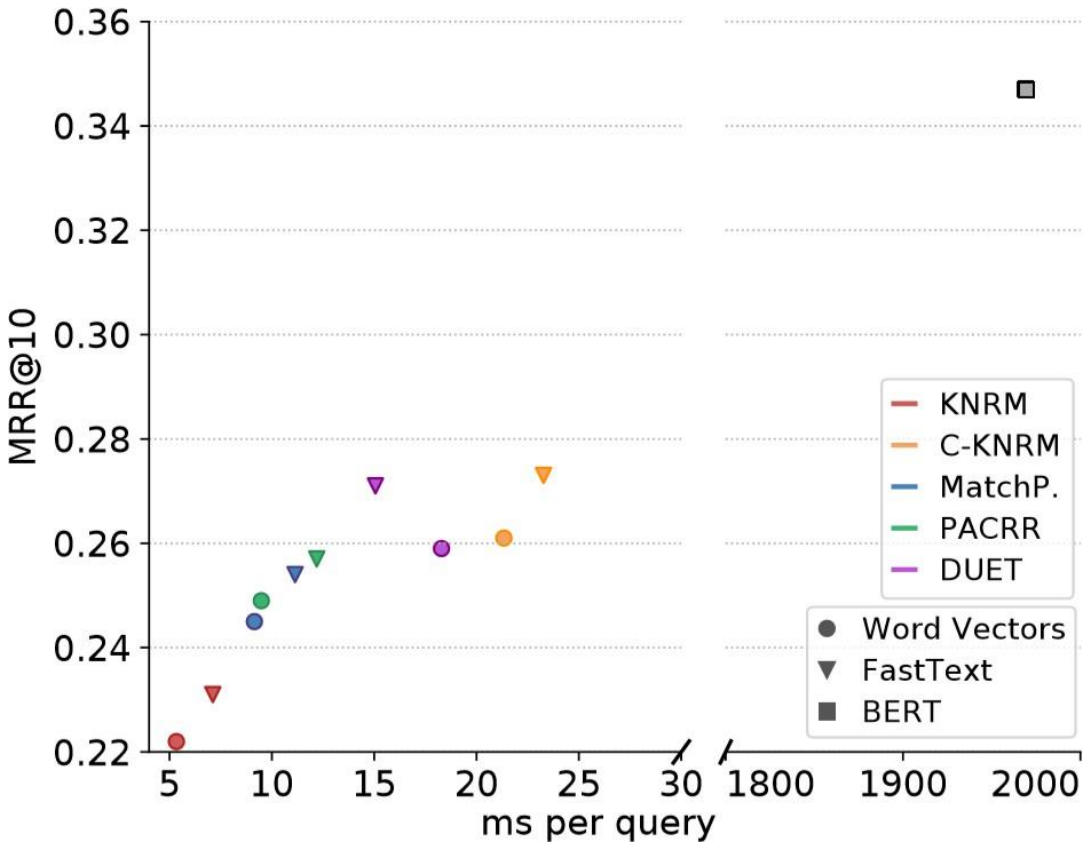
The Expando-Mono-Duo Design Pattern for Text Ranking with Pretrained Sequence-to-Sequence Models
Ronak Pradeep, Rodrigo Nogueira, Jimmy Lin https://arxiv.org/abs/2101.05667

$BERT_{CAT}$ for Longer Documents

- BERT is capped at max. 512 input tokens (query + document)
- Simplest solution: just cap the document at 512-query length
  - Works surprisingly well already (for MSMARCO-Documents)
  - But might not work well in other domains, where documents are really long, or contain a variety of topics at different depths
- Still simple, but working on full documents: Sliding window over the document -> take max window score as document score
  - Now, we can also make smaller sliding windows
    - Might be useful to use in the UI -> highlight the most relevant passage as snippet

Deeper Text Understanding for IR with Contextual Neural Language Modeling. Zhuyun Dai, Jamie Callan
https://arxiv.org/abs/1905.09217

$BERT_{CAT}$ In-Efficiency



- Evaluated on 250 docs / query on short MSMARCO-Passage *(max 200 tokens)*
- Basic IR-specific networks are fast, but moderately effective
- Transformer-based BERT is very effective, but very slow
  - + Infrastructure cost (blocking 1 GPU for 2 seconds at a time)

Sebastian Hofstätter and Allan Hanbury. 2019.
Let's measure runtime! Extending the IR replicability infrastructure to include performance aspects . In OSIRRC @ SIGIR.

Achieving Efficiency

- Multiple paths to reduce query latency
  - Query latency is our focus today, but full lifecycle efficiency also a concern
  - Lifecycle efficiency includes training, indexing and retrieval steps

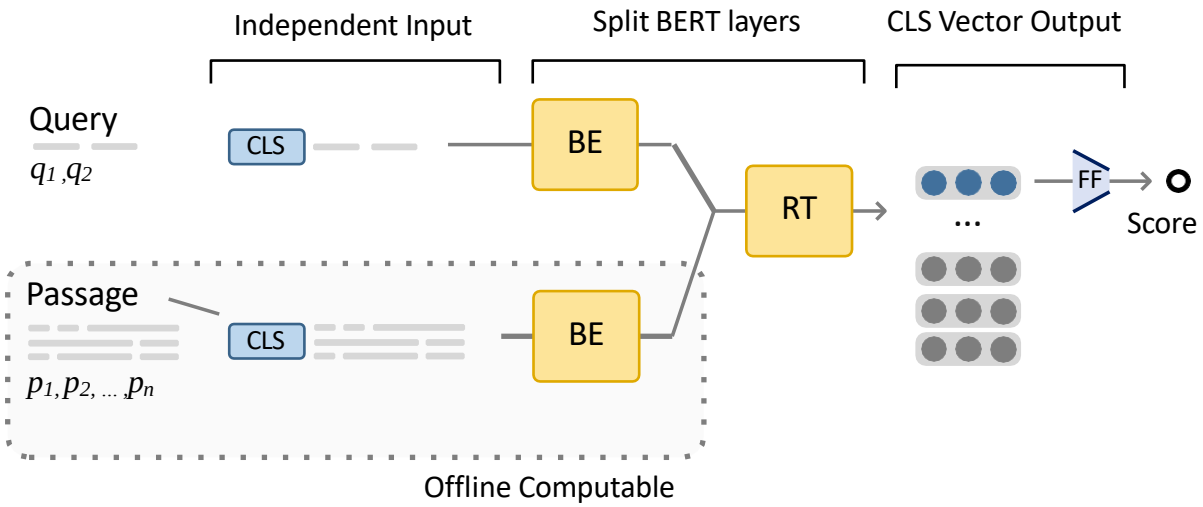
❶ Reduce model size
  - Smaller models run faster, duh!
  - Only possible until a certain threshold after which quality reduces drastically

❷ Move computation away from query-time
  - Pre-compute passage representation, so they become a simple lookup
  - Lightweight aggregation, that can be done at query time

Splitting BERT for Efficiency

- $BERT_{CAT}$ needs x (re-ranking depth) full evaluations at query time
  - Bad for query latency, deep re-ranking implausible
- Most of that computation is spent on passage encoding
  - If we can move the passage encoding to the indexing phase
  - Encode each passage exactly 1 time
  - At query time only need to encode 1 query (with few words, so it's fast)
- Multiple approaches have been proposed to then *glue* the query and passage representations back together
  - With only a small reduction in effectiveness

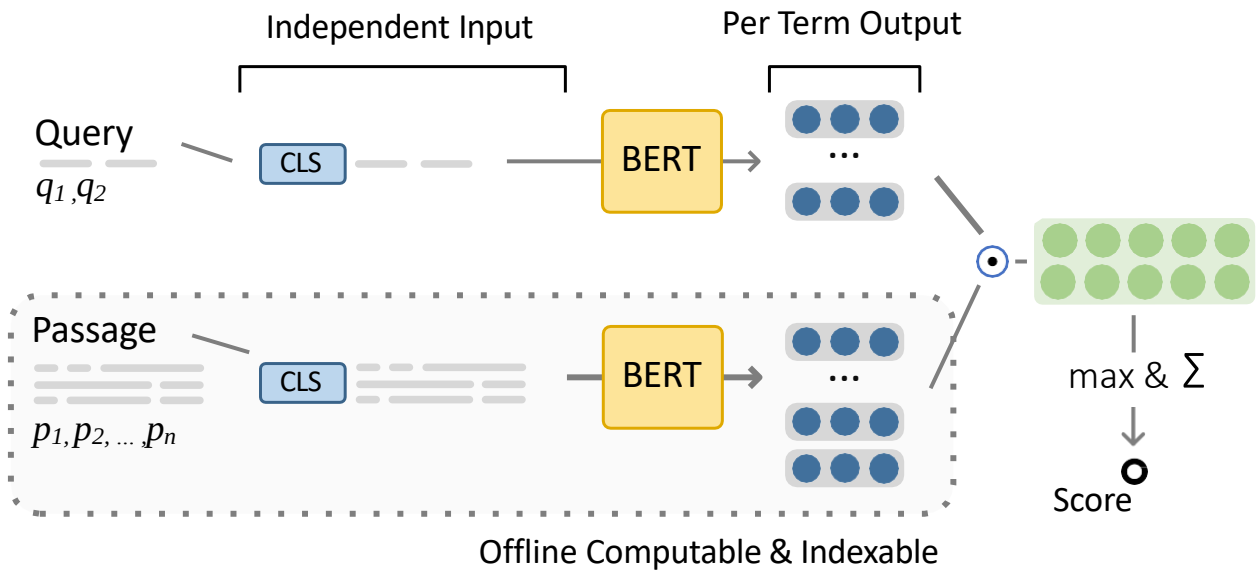
Splitting BERT: PreTTR



- PreTTR splits BERT layers:
  - The first *n* layers are separated
  - The following layers are concatenated again
  - *n* is a hyperparameter
- We can pre-compute the first-n layers of the passages
- Pro: ≈ quality as $BERT_{CAT}$
- Neg: Still low query latency + storage requirements

Efficient Document Re-Ranking for Transformers by Precomputing Term Representations; MacAvaney et al. SIGIR 2020 https://arxiv.org/abs/2004.14255

Splitting BERT: ColBERT



- ColBERT creates a match-matrix of BERT term-representations
- Then uses a simple max-pooling for the doc-dim & sum for the query dim
- Pro: Very fast query latency
- Con: Need to save all passage term vectors (huge storage cost)

ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT; Khattab et al. SIGIR 2020 https://arxiv.org/abs/2004.12832

ColBERT

- Simple formula (as long as we abstract BERT):

Encoding

$$\hat{q}_{1..n} = BERT([CLS]; q_{1..n})$$

$$\hat{p}_{1..n} = BERT([CLS]; p_{1..n})$$

Can be done at indexing time

Aggregation

$$s = \sum_{i=1..n} \max_{t=1..m} \hat{q}_i \cdot \hat{p}_t$$

Quite efficient at query time

$q_{1..n}$	Query tokens
$p_{1..m}$	Passage tokens
BERT	Pre-trained BERT model
[CLS]	Special tokens
s	Output score

Summary: Neural Re-Ranking for IR

1. Word level match matrix: core building block of Neural IR models

2. $BERT_{CAT}$: Slow but effective

3. Large Efficiency Gains with more pre-computing
(Pre-TTR, ColBERT)

Did we Reach the State-of-the-art?

Almost!

- Current research is heavily based on LLMs:
  - IR for LLM: Retrieval-Augmented Generation
  - LLM for IR: e.g. query expansion, reformulation…

- Many questions are open
  - Context-awareness, Explainability, Generalization across domains…

- General direction: More computation = better results,
  possible by better hardware & more data

More Resources

- Pretrained Transformers for Text Ranking: BERT and Beyond
  Jimmy Lin et al
  https://arxiv.org/abs/2010.06467
  - Great Survey on different techniques using Transformers in Ranking

- Large Language Models for Information Retrieval: A Survey
  https://arxiv.org/abs/2308.07107
  - Survey on using LLMs

- Running list of new papers in the field: https://arxiv.org/list/cs.IR/recent
  - ~20 new papers a day; the fastest source to get info about pre-prints

Next Stop:

Large Language Models