Deep Retrieval, Reranking, and Ranking

CS6101

Autumn 2025

Map

- Started with sparse retrieval
- Studied context-invariant and then context-dependent word embeddings
- Aggregated these into passage embeddings
  - Or passage-pair embeddings
- Studied dense approximate nearest neighbor search
- Now, will apply these building blocks to dense (passage) retrieval (or ranking) aka DPR

Next…

- Late interaction single vector retrieval
  - LSH for various distance/similarity scores
  - Adaptive LSH
- Other ANN techniques (HNSW)
- Trainable (dense) retriever
  - aka “dense passage retrieval” or DPR (Haystack)
- Late interaction multi-vector retrieval
  - Middle ground between single-vector and early interaction
- Generative rerankers
- Generative retrievers (DSI)

Corpus adaptive LSH

- N docs from domain $\mathcal{X}$, LSH $h: \mathcal{X} \to \{-1,1\}^K$
- Replace h with neural network $f: \mathcal{X} \to [-1,1]^K$
  - E.g., ending with tanh activation
- What should a good $N \times K$ hash matrix look like?
- No sitting on the fence (no entry near zero)
  - $\sum_{n,k} \left| |f(x_n)[k]| - 1 \right|$ small: all $f(x_n)[k] \to \pm 1$
- Each hash bit should evenly split the corpus (bit balance)
  - For each k, want $\sum_n f(x_n)[k] = 0$
- Different hash bits (columns) should be decorrelated
  - For $k \neq k'$, want $\sum_n f(x_n)[k] f(x_n)[k'] = 0$

Further informed by query workload

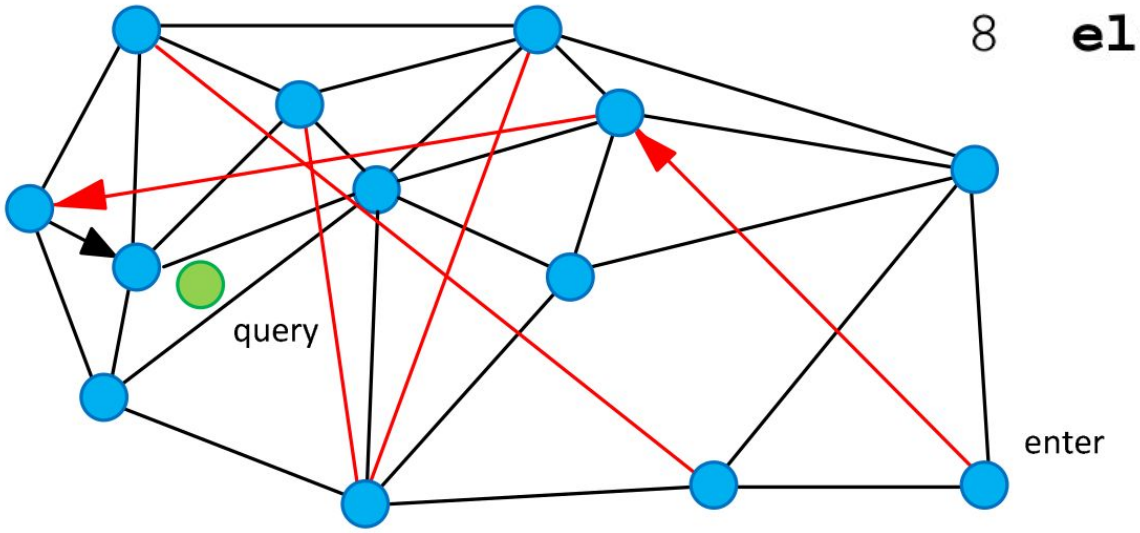
- Docs never touched by query workload can go anywhere
  - If they do not overpopulate frequently touched buckets
- For each q, get $N/2^K$ docs $\{d_\dagger\}$ with best scores
  - Use scoring function (dot, cos, L2, etc.) on whole corpus
  - Or use relevant docs if known
- Want these docs in bucket $h(q)$, i.e. $f(q) = f(d_\dagger)$
  - Suggests loss $\sum_{d_\dagger} \sum_k (1 - f(q)[k]\ f(d_\dagger)[k])$
  - (Other forms possible, may behave better)

Additional trick: quantization

- Original vector x_i quantized to $\tilde{x}_i$
- May want to minimize $\mathbb{E}_q \left[\sum_i (\langle q, x_i \rangle - \langle q, \tilde{x}_i \rangle)^2 \right]$
- Equals $\mathbb{E}_q \left[\sum_i \langle q, x_i - \tilde{x}_i \rangle^2 \right]$
- Assume $\mathbb{E}_q[qq^T] = c\mathbb{I}$ (query vectors isotropic)
- Objective proportional to $\sum_i \|x_i - \tilde{x}_i\|^2$
- Not all (x, q) pairs are equally important
- Approximation error on pairs with large $\langle q, x_i \rangle$ more serious
  - $\longrightarrow$ score-aware quantization loss
- $\ell(x_i, \tilde{x}_i, w) = \mathbb{E}_q \left[w(\langle q, x_i \rangle) \langle q, x_i - \tilde{x}_i \rangle^2 \right]$

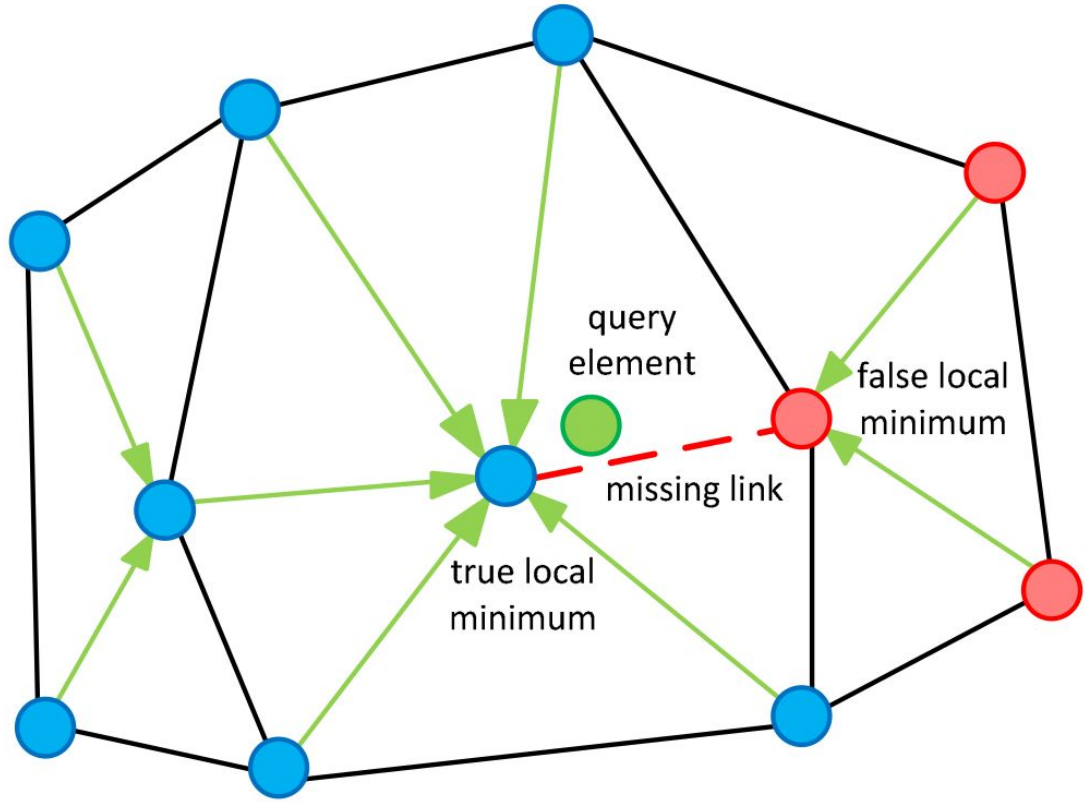
(Hierarchical) network small world

```
Greedy_Search(q: object, v_enter_point: object)
1   v_curr ← v_enter_point;
2   σ_min ← σ(q, v_curr); v_next ← NIL;
3   foreach v_friend ∈ v_curr.getFriends() do
4       if σ_fr ← σ(query, v_friend) < σ_min then
5           σ_min ← σ_fr;
6           v_next ← v_friend;
7   if v_next = Nil then return v_curr;
8   else return Greedy_Search(q, v_next);
```



HNSW multi-search

```
Multi_Search(object q, integer: m)
1  results: SET[objects];
2  for (i ← 0; i < m; i++) do
3     entry_point ← getRandomEntryPoint();
4     local_min ← Greedy_Search(query, entry_point)
5     if local_min ∉ results then
6        results.add(result);
7  return results;
```

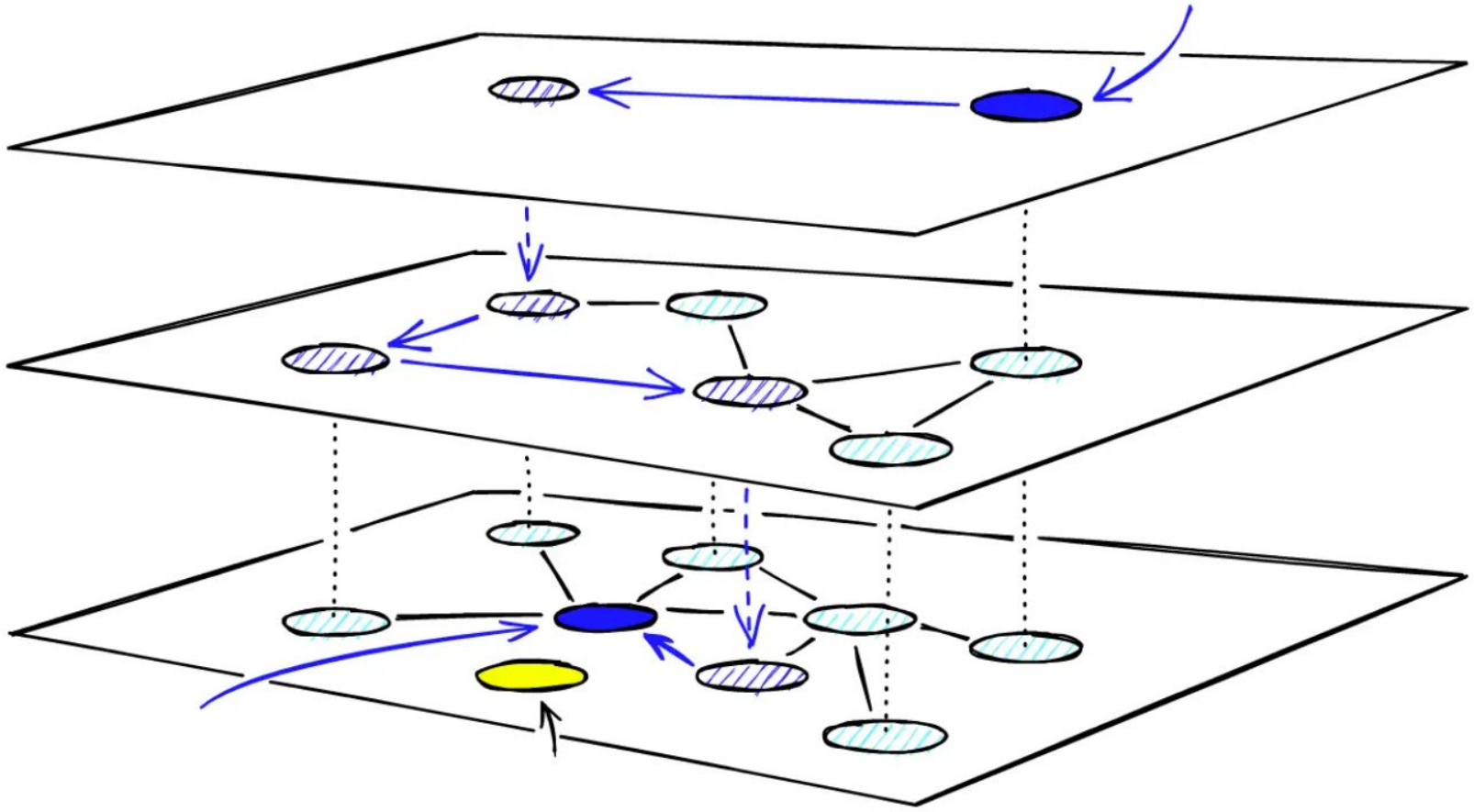


HNSW new item insertion

```
Nearest_Neighbor_Add(object: new_object, integer: k, integer: w)
1  SET[object]: localMins ← Multi_Search (new_object, w);
2  SET[object]: u ←∅; //neighborhood;
3  foreach object: local_min ∈ localMins do
4    u ← u ∪ local_min.getFriends();
6  sort the set u so to satisfy the condition σ (u[i],
new_object) < σ (u[i+1], new_object)
7  for (I ← 0; i < k; i++) do
8    u[i].connect(new_object);
9    new_object.connect(u[i]);
```

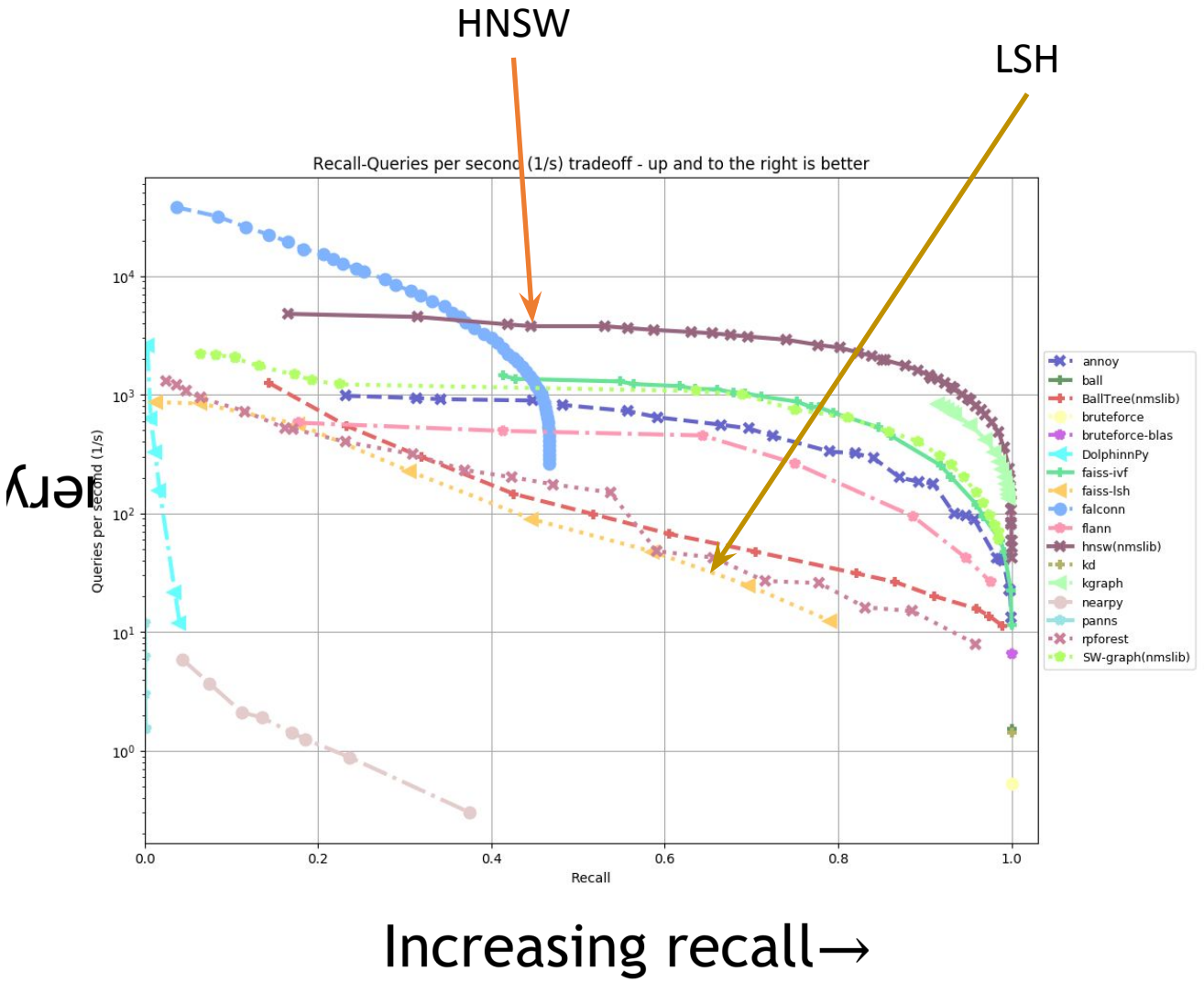
- Entirely heuristic compared to LSH, but quite successful in practice
- Later, a hierarchical version (hence, HNSW)

Hierarchy



HNSW vs LSH comparison

- HNSW seems to perform much better than faiss-lsh
  - LSH implementation in FAISS package
  - L2 and dot
- Is faiss-lsh data-dependent or agnostic?



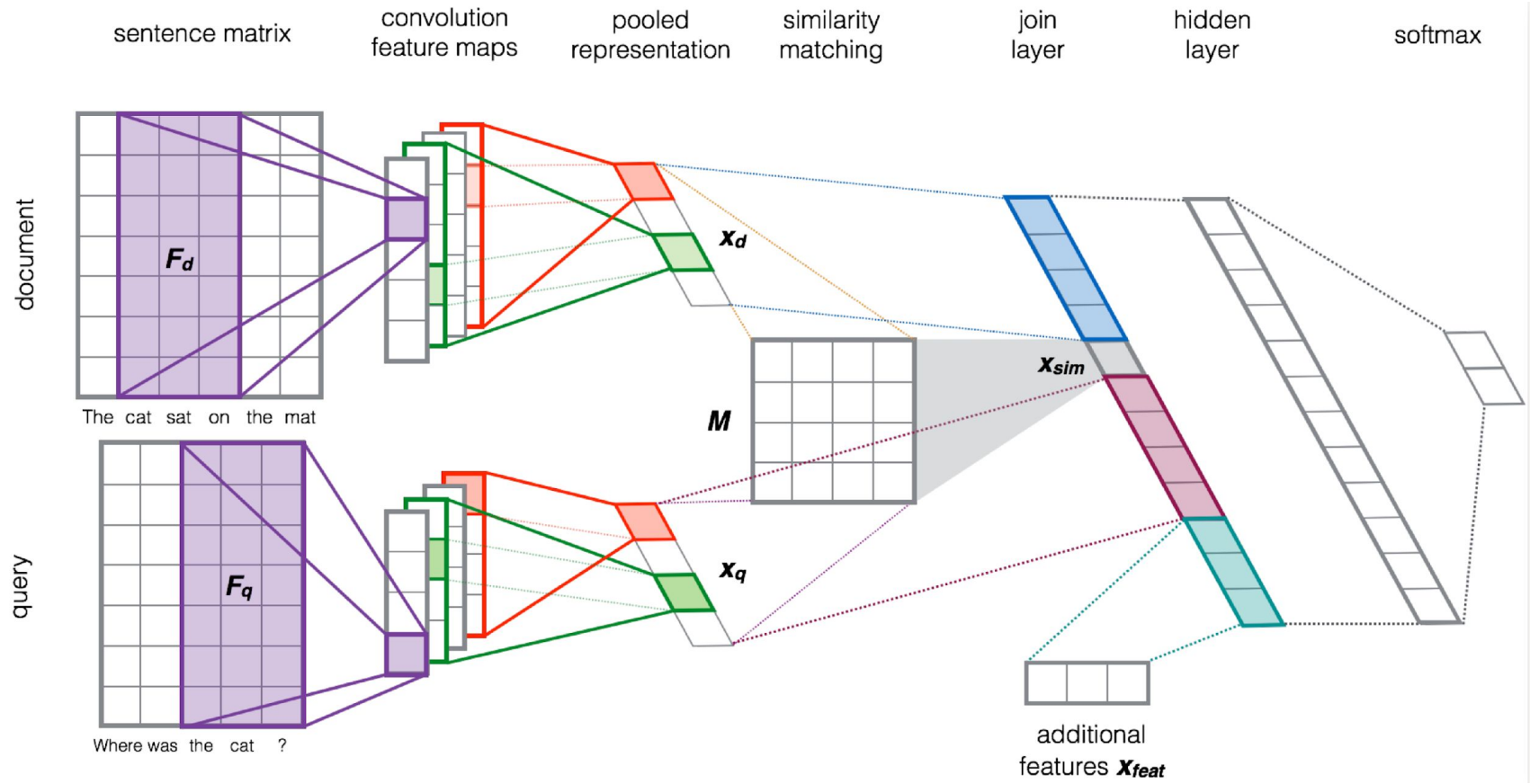
Deep retrieval data set

- MSMARCO https://microsoft.github.io/msmarco/
- 100,000 real Bing questions
- Human generated answer
- Tasks
  - Document ranking leaderboard
  - Passage retrieval leaderboard
  - Also QA and NL generation
- MSMARCO is contained in the BEIR benchmark
  - Heteregenous benchmark with diverse IR tasks
  - Challenging out-of-domain IR evaluation
- LoTTE: another out-of-domain eval benchmark

References

- Huang+2013 bag-of-word-embeddings, symmetric aggregation, dot product similarity
- Shen+2014 use convnet instead of bag
- Palangi+2015 use LSTM instead of convnet
- Gillick+2018 avg word vectors; improved sampling and loss
- Bi-encoder vs cross-encoder
- Karpukhin+2020 difficult negatives from BM25
- Xiong+2020 difficult negatives from model itself
- Humeau+2019 Khattab+2020 address cross-encoder inefficiency

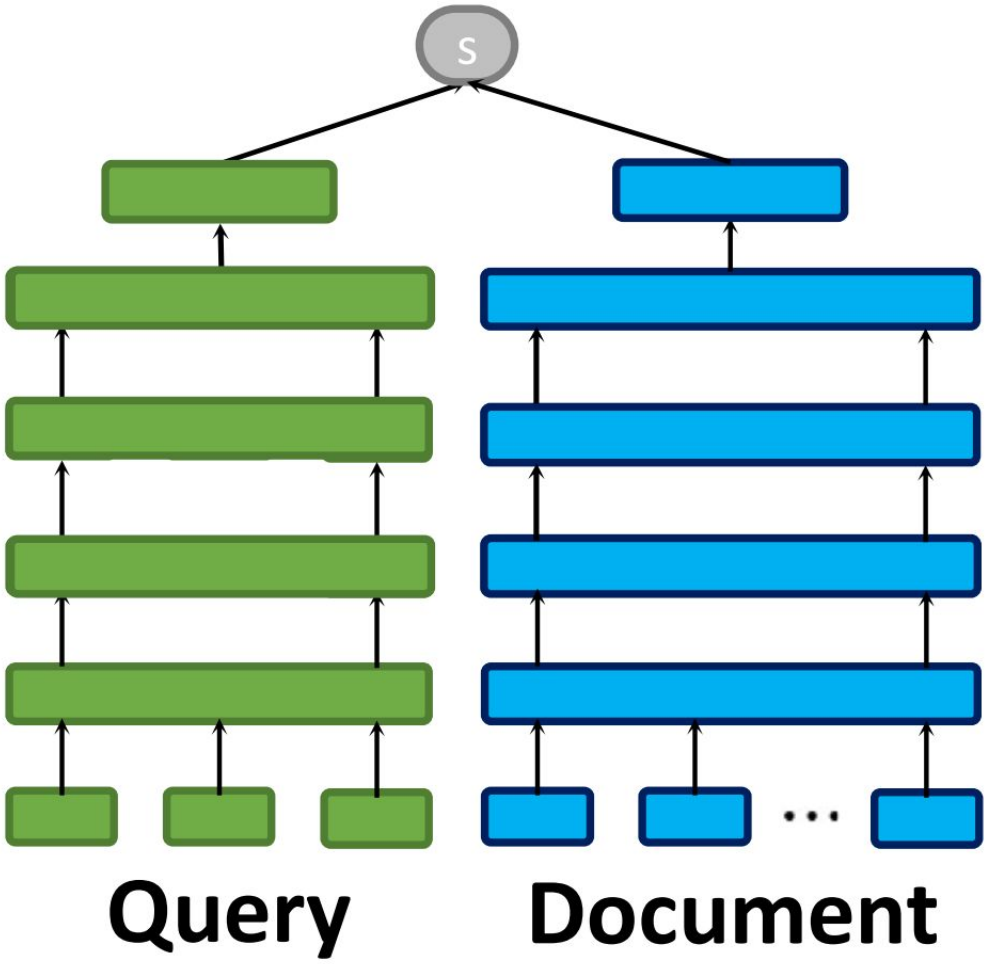
Comparing query and passage



If there are “cross features” or the output network is complicated, may not be able to do efficient top-K retrieval

- Process query and passage through ‘Siamese’ (twin) networks with tied model weights
- Results in vectors x_d, x_q
- Compare using another output network to predict relevance score

Deep retrieval: late interaction

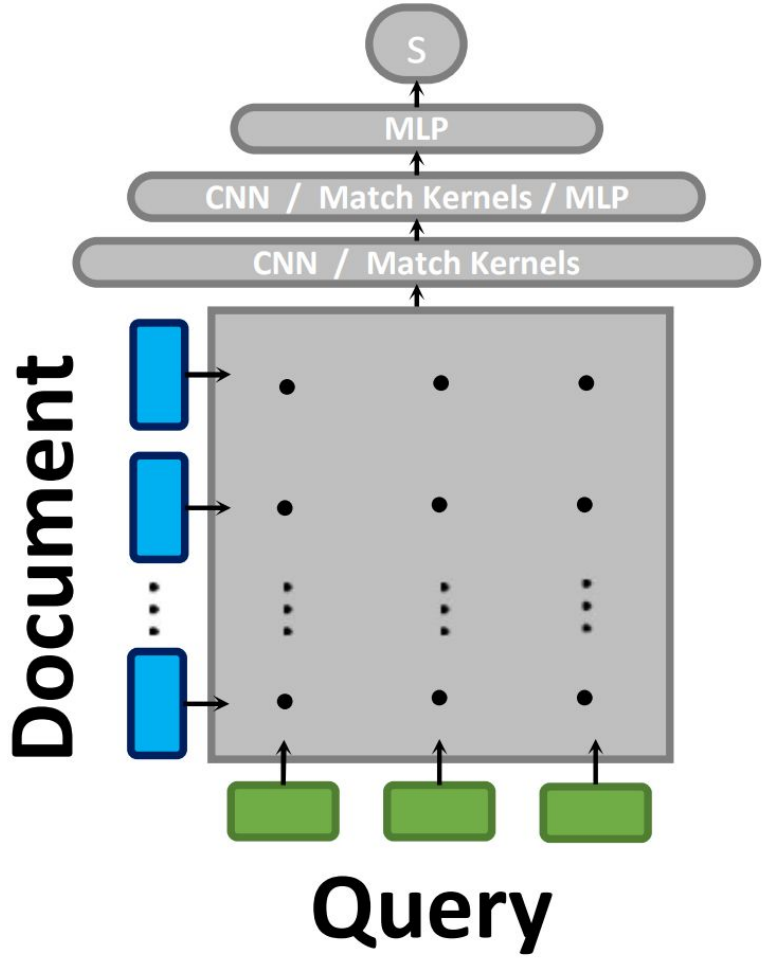


(a) Representation-based Similarity

(e.g., DSSM, SNRM)

- Too much compression
- Fixed size rep of query and doc does not support best scoring/ranking
- Fast similarity search possible via LSH/ANN

Deep retrieval: crossbar

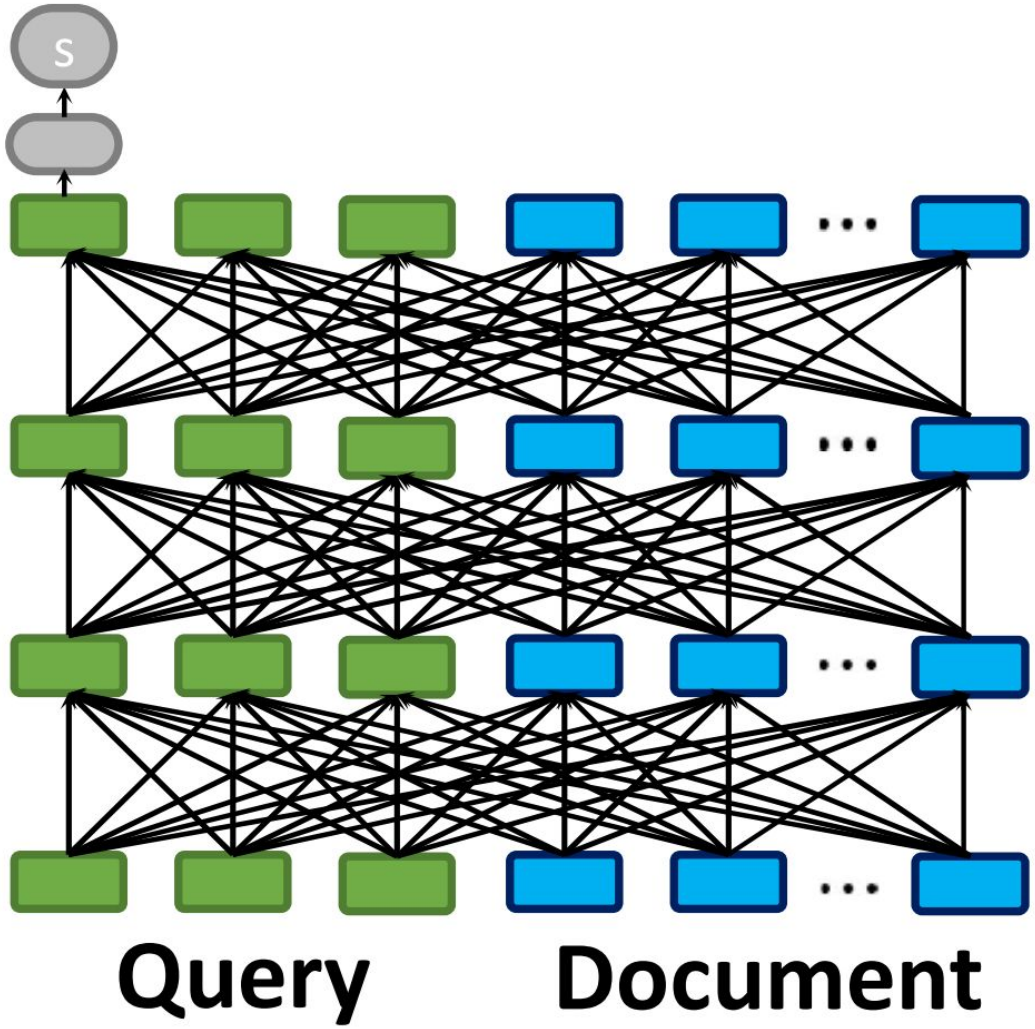


(b) Query-Document Interaction

(e.g., DRMM, KNRM, Conv-KNRM)

- All (promising) docs to be checked; slow
- No self-attention

All-to-all early interaction



(c) All-to-all Interaction

(e.g., BERT)

- Enables contextualize within and across the two passages
- Even slower
- Soft vs hard matches?
- Position embeddings, layout tricks

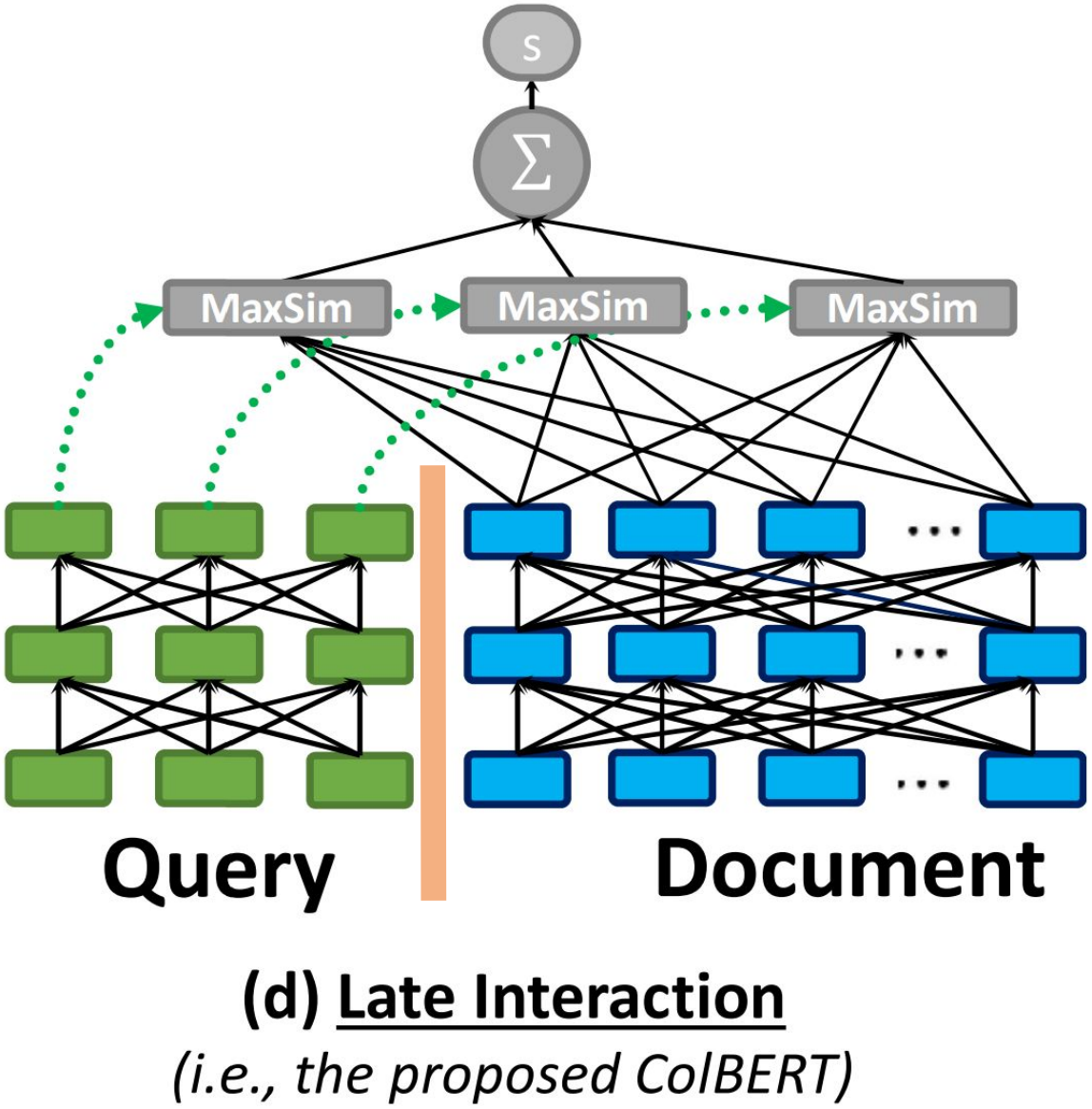
Model distillation

- Use slow early-interaction model (‘teacher’) to generate scores for relevant and irrelevant doc samples
- Train fast late-interaction model (‘student’) to replicate these scores
- Expect trained student model to be somewhere in between early and late interaction models
- Lot depends on sampling protocol

Another compromise between early and late

- Slow and accurate vs fast and not-so-accurate
- On the corpus side
  - Limit word vector contextualization to within passage
  - Do not depend on query
- Query word vectors contextualized at query time
- Do not collapse word vectors into fixed size question/passage vectors
- Implies multiple probes into ANN data structure

Late, sparse interaction (ColBERT)



(d) Late Interaction

(i.e., the proposed ColBERT)

- Separate stacks for q, d

$$E_q := \texttt{Normalize}(\ \texttt{CNN}(\ \texttt{BERT}(\text{“}[Q]q_0q_1\ldots q_l\#\#\ldots\#\text{”})\ )\ )$$

$$E_d := \texttt{Filter}(\ \texttt{Normalize}(\ \texttt{CNN}(\ \texttt{BERT}(\text{“}[D]d_0d_1\ldots d_n\text{”})\ )\ )\ )$$

- Scoring is not one-shot but sum over query tokens

$$s(d|q) = \sum_i \max_j E_q[i] \bullet E_d[j]$$

- Each q token picks best d cover

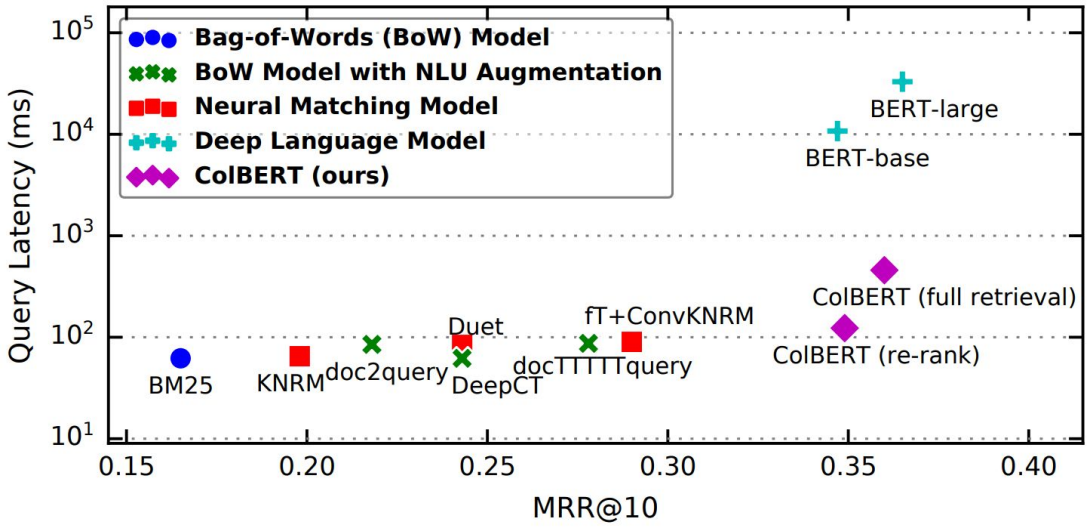
ColBERT (v2) implementation

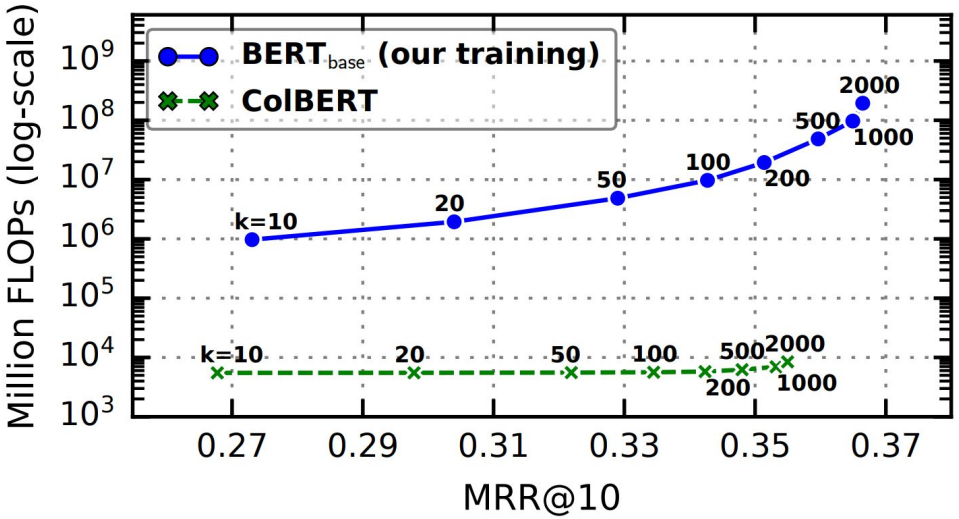
- Every contextual word embedding vector is unique
  - Cannot use "word → posting list" map as in sparse index
- Cluster these (kmeans, say); each cluster records
  - Centroid vector $\overline{\boldsymbol{w}}$, used to assign query word to cluster
  - List/set of postings, each containing a doc ID and embedding(s) $\boldsymbol{w}$ of word(s) in it that belong(s) to the cluster
  - Needed to compute impact scores
- But these word embeddings cost a lot of space
  - "Vectors corresponding to each sense of a word cluster closely, with only minor variation due to context"
  - Express $\boldsymbol{w} = \overline{\boldsymbol{w}} + \boldsymbol{\delta}$ and store $\boldsymbol{\delta}$ using 1-2 bit precision

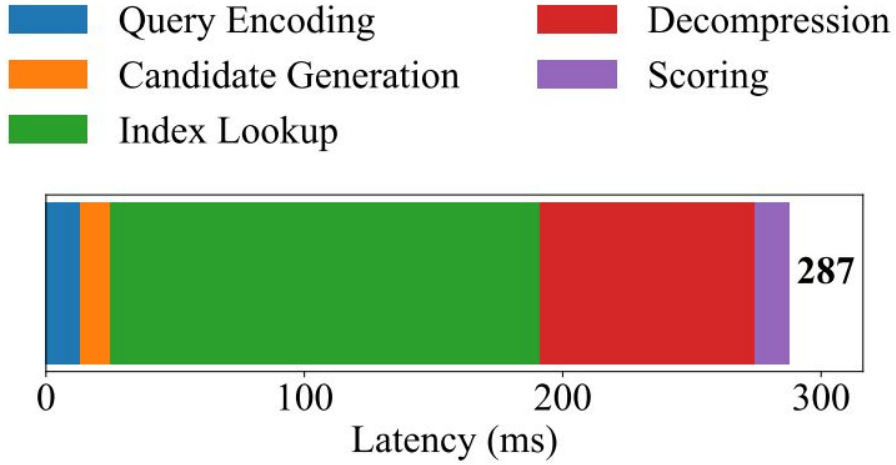
Supervision (for fine-tuning BERT)

- (No trainable weights in max-sim score)
- ColBERT v1 used official $\langle q, d_{\oplus} \rangle$ pairs
  - Can also use high-quality cross-encoder to get 'silver' $d_{\oplus}$
    - A form of model distillation
  - Triples formed by sampling $d_{\ominus}$ using BM25
  - Systems sensitive to negative sampling
- ColBERT v2 used v1 to get negative samples $d_{\ominus}$
- Batch has 64 passages, 1 positive

Performance summary







(a) Vanilla ColBERTv2 (nprobe=4, ncandidates=2^{16}).

Why is index lookup taking such a long time?

What is decompression?

How to save index lookup and decompression time? PLAID, PLAIDv2

Colbert → PLAID pruning heuristics

- Three selection and pruning stages
- Collect centroids close to each query word, union
- Prune centroids not well supported by at least one query word
- Score documents using centroids and prune some
- Fully score surviving documents by decompressing centroid + delta

Back to sparse retrieval

- Pendulum swung from sparse/‘lexical’ to dense index
- … then swung back a bit toward ‘lexical’ inv index
- Continue making use of dense contextual word vectors
- But make each doc and query vector sparse(r)
- Allowing return to(ward) inverted index
  - Zamani+2018 (SNRM): among the earliest proposal
  - Paria+2000 (FLOPS): improved regularization
  - SPLADE

SNRM Zamani+2018

- "Standalone neural ranking model"
  - As against re-ranking after fast filtering
  - Potential to mitigate filter-stage loss of recall
- Focus on late interaction scoring function

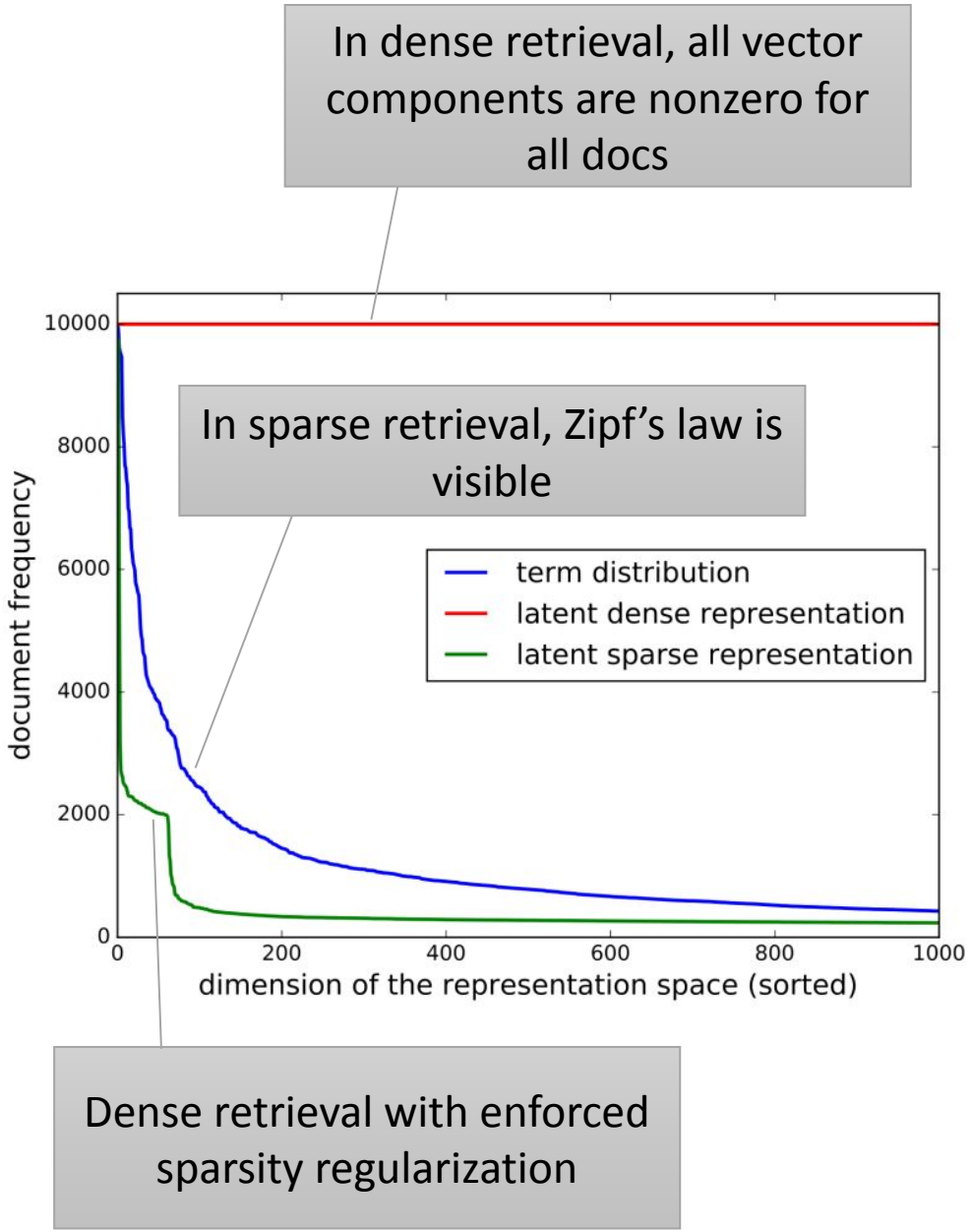
$$\text{score}(q, d) = \psi\big(\phi_Q(q), \phi_D(d)\big)$$

  - ϕ_D runs when preprocessing the corpus
  - Outputs high-dimensional sparse vector on latent features
  - ϕ_Q and ψ run (fast) at query time
- As a 2018 paper, they used $\phi_D(d) = \sum_{i=1}^{|d|-n+1} \phi_{\text{ngram}}(d_{i:i+n-1})$
- Today, given enough GPUs, we can use a transformer encoder
- Standard triplet loss $\left[\delta + \psi\big(\phi_Q(q), \phi_D(d_\oplus)\big) - \psi\big(\phi_Q(q), \phi_D(d_\ominus)\big)\right]_+$
- So far, nothing new

L1 regularization for sparsity

- We want $\phi_Q(q), \phi_D(d)$ to be sparse
- Technically the L_0 norm $\|\phi(x)\|_0$ which is not continuous
- Replace with closest convex surrogate $\|\phi(x)\|_1$
  - Called "Lasso regularization"
  - Widely used in sparse regression
- Overall loss for one triplet is

$$\mathcal{L}_{\text{triplet}}(q, d_\oplus, d_\ominus) + \lambda\left(L_1(\phi(q)) + L_1(\phi(d_\oplus)) + L_1(\phi(d_\ominus))\right)$$



From L1 to L2 regularization (FLOPS)

- Slight notation change: $f_\theta(x_i) \in \mathbb{R}^d$ is the sparse embedding of doc number i in corpus D of n docs
- Let $\overline{p}_j = \frac{1}{n}\sum_{1\le i\le n} [\![f_\theta(x_i)[j] \neq 0]\!]$ be the "fill fraction" of column j
- In the absence of query workload stats, assuming uniform probability on all docs, number of FLOPs

$$F_\theta(q, D) = \sum_{1\le i\le n} \sum_{j\in[d]} [\![f_\theta(q)[j] \neq 0]\!] [\![f_\theta(x_i)[j] \neq 0]\!]$$

- With some more assumptions, can show

$$\mathbb{E}_{q\sim D}[F(q, D)] \propto \sum_{j\in[d]} p_j^2$$

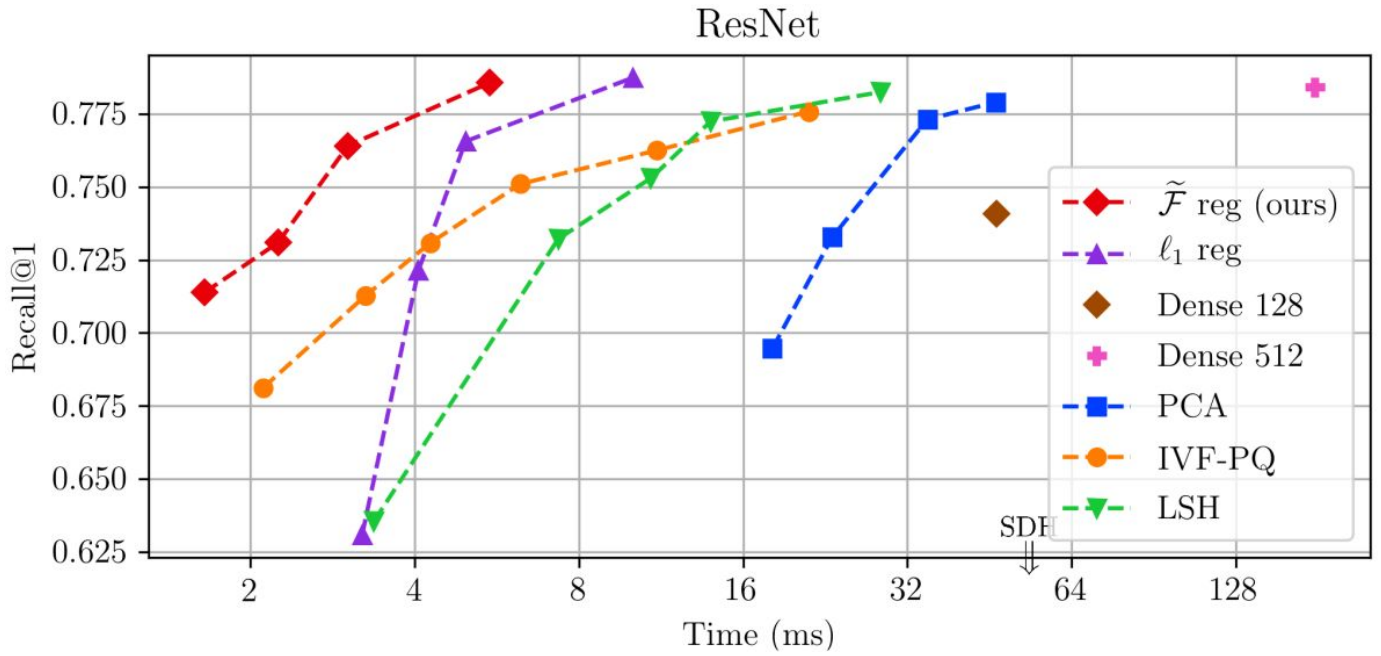
- But what happens to sparsity? (L2 discourages sparsity)

FLOPS objective

- Let $\overline{a}_j = \frac{1}{n}\sum_{i\in[n]} |f_\theta(x_i)[j]|$
- Objective is to minimize

$$\mathcal{L}_{\text{triplet}}(q, d_\oplus, d_\ominus) + \lambda \sum_{j\in[d]} \overline{a}_j^2$$

- How about $\sum_{j\in[d]} \left(\frac{1}{n}\sum_{i\in[n]} [|f_\theta(x_i)[j]| - T]_+\right)^2$?
  - T is a threshold hyperparameter to tune sparsity
  - This is how a real system would sparsify the inverted index



SparTerm, SPLADE and SPLADEv2

- BERT has base embedding matrix E
- E_j is the jth row, base embedding of word indexed j
  - j can be any word in corpus vocabulary
- Text $t = (t_i : i \in [n])$ can be query or passage
- Pass t through BERT to get $(h_i : i \in [n])$
- Similarity of word j with contextual embedding h_i designed as $w_{i,j} = \text{transform}(h_i) \bullet E_j + b_j$
  - transform = linear layer, GELU activation, layer norm
  - b_j is a bias term specific to word j

What role is transform(…) supposed to fulfill?

How strongly does text t 'activate' term j?

- Aggregate $w_{i,j}$ over all text tokens (indexed i) to get

$$w_j = \sum_{i \in [n]} \log\left(1 + \mathrm{ReLU}(w_{i,j})\right)$$

- In v2, modified to

$$w_j = \max_{i \in [n]} \log\left(1 + \mathrm{ReLU}(w_{i,j})\right) \geq 0$$

- Bias b_j chosen/trained so most $w_j = 0$
- Text t represented as vector $(w_j : j \in [d])$
- Query and doc texts compared via dot product
  - Call this raw score $s(q, d) \geq 0$

Relevance loss term

- Somewhat different from triplet hinge loss
- Hard positive and negative $\langle q, d_{q\oplus}, d_{q\ominus} \rangle$
- Add in-batch 'silver' negatives $D_{q\boxminus}$
- $$\mathcal{C}_{\text{rank\_IBN}} = \frac{\exp\left(s(q, d_{q\oplus})\right)}{\exp\left(s(q, d_{q\oplus})\right) + \exp\left(s(q, d_{q\ominus})\right) + \sum_{d_{q\boxminus} \in D_{q\boxminus}} \exp\left(s(q, d_{q\boxminus})\right)}$$
- A form of logistic loss

Regularizer loss term (SPLADE)

- Regularizer remains same as FLOPS
  - There are C texts $t^{(c)}$ where $c \in [C]$
  - Extend notation w_j to $w_{c,j}$ = activation of term j by text indexed c
- $\mathcal{C}_{\text{FLOPS}} = \sum_{j \in [d]} \left(\frac{1}{C} \sum_{c \in [C]} w_{c,j} \right)^2$
- More standardized evaluation (also check paper)
  - Do ablations reveal effects of each mod separately?
  - Are ColBERTv2/PLAIDv2, SNRM, FLOPS, SPLADEv2 compared under carefully controlled policies and settings?

Finally … the brave new world of LLMs

- Given per-token time and cost, almost always for reranking
- May fine-tune (smaller LMs) for retrieval
- Two styles
  - LLM ‘prompt’ includes query and one passage to be scored Nogueira+2019
  - LLM ‘prompt’ includes instruction, query and several passages to be ordered by relevance Reddy+2024

(q, d) to [CLS] embedding to relevance

- Nogueira+2019
  - Expensive cross-interaction between query and candidate
  - "[CLS] q a" → Enc → [CLS] embedding → relevance score s_i
  - Maximize $\sum_{i \in D_{q\oplus}} \log s_i + \sum_{i \in D_{q\ominus}} \log (1 - s_i)$
- Huge leap beyond sparse retrieval

Method	MSMARCO MRR@10	
	Dev	Eval
BM25 Lucene	16.7	16.5
BERT-base	34.7	
BERT-large	36.5	35.8

More tweaks — Reddy+2024 (FIRST)

- Why not just ask LLM to rerank passages?
  - LLM may mutilate, drop or hallucinate passages
  - Prevent by assigning passage IDs (may be special tokens)
- Do not decode relevance-ordered passage ID sequence
- Instead, find output distribution limited to passage ID tokens
- Use probabilities from this distribution to order IDs

Generative retrieval

- "Differentiable search index" (DSI) Tay+2022
- Finetune LLM as seq2seq to convert query to doc IDs
  - Base LLM used = T5
  - Effectively, use LLM model weights to store 'index'
- A doc ID could be an opaque string or a short readable token sequence
- Training or fine-tuning losses driven by these tasks
  - Doc tokens → doc ID
  - Doc ID → doc tokens
  - Roundtrip Doc tokens → doc ID → doc tokens
- First 32 or 64 "doc tokens" best (better than full doc)

Doc ID representation for DSI

- Unstructured atomic (integer) identifiers
  - Have to extend LM output vocabulary
- Natively structured (tokenizable) string identifiers
  - Can decode one ID token at a time using beam search
  - Shared representation for shared ID tokens
- Semantically structured identifiers
  - Hierarchical clustering of (single-vector) doc embeddings from 8-layer BERT encoder
  - Form a (decimal) trie; paths like 0.3.5.1.99 (leaf has 100 docs)

Concluding remarks

- **Gen0:** sparse inverted index, cosine, dot, BM25
- **Gen1:** context-independent word embeddings, simple aggregations over question and passage, late interaction through single-vector probe
- **Gen2:** contextual word embeddings, more complicated aggregation, or early interaction
- **Gen3:** late interaction through multiple ANN probes
- **Gen4:** very high dim, very sparse query and doc rep, elements can be uninterpretable or reuse term space
- **Gen5:** generative retrieval: query → 'identifiers'