An Efficiency Study for SPLADE models

Carlos LASSANCE, Stéphane CLINCHANT
Introduction

• Goal: Study the efficiency of SPLADE models for sparse neural retrieval
  • In domain: MSMARCO passage dataset
  • Out-of-domain: 18 BEIR datasets
Introduction – SPLADE Recap

• Use the MLM output
  • Max pooling over each token output
  • Induce sparsity:
    • ReLU over the output
    • FLOPS [Paria et al 2020] regularization
      • Estimate number of activations in a batch
      • Proxy for total retrieval FLOPS

http://github.com/naver/splade
Motivation: Findings from Wacky Weights
Mackenzie, Trotman and Lin, 2021

Findings:
• Recent sparse models are slower than BM25
• SPLADE 50x slower on mono-thread evaluation

RQ:
SPLADE quick as BM25?
First things first: 
Is SPLADE efficient?

• **Yes and No**
  - **No:** It does not optimize for the same things as sparse retrieval
    - Released models are *tuned for effectiveness*, not efficiency
    - Optimized *for multi-thread retrieval* of each query
      - Measures FLOPS, not latency
  
• **Yes:** *SPLADE is a family of models*
  - Control efficiency-effectiveness trade-off
  - Can optimize for cpu mono-thread query retrieval:
    - Focus more on query size than document size
Finding efficient SPLADE configurations

- I) Explore SPLADE family to find better configuration
  - Small, Medium and Large versions
- II) Use latest available data (better distillation)
Our contribution

• Can we go further than those adjustments?

• III) Separating encoders
  • Traditional SPLADE makes no difference between query and document
  • Hard for the model to learn that sparsities may be different

• IV) Using L1 regularization instead of FLOPS on queries
  • FLOPS is optimized for generating balanced indexes
  • Queries need to be small, but don’t need to be balanced

• V) Unsupervised FLOPS+MLM training
  • Improves the state of the network before pretraining
  • Network already knows output should be sparse
Results: Improvements add up
VI) Reducing query encoder latency

- VI-BT) Using a smaller query encoder (BERT-Tiny)
  - Reduces the query encoder latency to almost 0 (43 ms -> 0.7ms)

- VI-SD) SPLADE doc
  - No encoding
  - *: without stop words
Comparison with SoTA sparse on in-domain data (MSMARCO)
Comparison on OOD (BEIR)

| Method                   | Latency | MSMARCO | TREC19 | BEIR | BEIR* |
|--------------------------|---------|---------|--------|------|-------|
| Baselines                |         |         |        |      |       |
| BM25†                    | 4       | 19.7    | 50.6   | 43.0 | -     |
| DocT5 [36]               | 11      | 27.6    | 64.2   | 44.1 | -     |
| SPLADEv2-distil [10]     | 691     | 36.8    | 72.9   | 47.0 | 49.3  |
| Proposed models          |         |         |        |      |       |
| VI) BT-SPLADE-S          | 7       | 35.8    | 67.2   | 39.2 | 45.9  |
| VI) BT-SPLADE-M          | 13      | 37.6    | 69.4   | 42.1 | 47.1  |
| VI) BT-SPLADE-L          | 32      | 38.0    | 70.3   | 44.5 | 48.0  |

BEIR* creates an ensemble with BM25 to non BM25-baselines
Latency increases by 4 ms
Comparison with dense models

*How to?*

- Not exactly sure how to do it fairly
  - Different software makes for different benchmark
    - Comparing PISA/Anserini/JASS vs NMSlib/FAISS?
    - Example: How to be sure that all of them are warmed up correctly/fairly?

- Different optimizations
  - Approximate KNN (Dense) vs KNN (Sparse)
  - “Uniform” Latency (Dense) vs “Variable” Latency (Sparse)
  - Mono-cpu (Latency) vs Multi-cpu/gpu (QPS)
  - Keep index small (IVF, PISA) vs Precompute and store everything (HNSW)
Comparison with dense models

*How to?*

**OPEN QUESTION**

Take results with a grain of salt

- Not exactly sure how to do it fairly
  - Different software makes for different benchmark
    - Comparing PISA/Anserini/JASS vs NMSlib/FAISS?
    - Example: How to be sure that all of them are warmed up correctly/fairly?

- Different optimizations
  - Approximate KNN (Dense) vs KNN (Sparse)
  - “Uniform” Latency (Dense) vs “Variable” Latency (Sparse)
  - Mono-cpu (Latency) vs Multi-cpu/gpu (QPS)
  - Keep index small (IVF, PISA) vs Precompute and store everything (HNSW)
Comparison with dense models

Latency

OPEN QUESTION
Take results with a grain of salt
Comparison with dense models

**QPS**

**OPEN QUESTION**
Take results with a grain of salt
Conclusion

SPLADE can be efficient and VI) BT-Medium is the first method to concurrently:

- Only 2x the cost of BM25 (or 4 times of BM25 without stop words)
- Comparable to ColBERTv2 on MSMARCO (<10% loss of MRR@10)
- Comparable to SPLADEv2 on BEIR (<5% loss of NDCG@10)

Code: [https://github.com/naver/splade](https://github.com/naver/splade)
Indexes: [https://github.com/naver/splade/tree/main/efficient_splade_pisa](https://github.com/naver/splade/tree/main/efficient_splade_pisa)
HuggingFace weights: [https://huggingface.co/naver](https://huggingface.co/naver)
Improving other sparse methods

• Kinda unfair comparison with them as well
• Distillation and hyperparameter search can easily be added to both
• Better PLM initialization as well
  • MLM+Flops? Contriever? CoCondenser?
• Removing stop words from queries could also be important
• Is there a way to benchmark all this?