Condenser and Adaptive Batch Scheduling for OpenQA Passage Retrieval

Zikun Cui,[1] Xiaoying Yang,[2] Yiting Zhao[3]

[1] Department of Civil and Environmental Engineering, Stanford University
[2]Department of Computer Science, Stanford University
[2]Department of Management Science & Engineering, Stanford University

Problem

Open-domain question answering(OpenQA)

- Find answers for given questions from a large collection of documents

Current Solution

- Pipeline
  - Retrieve: search for relevant documents to the question
  - Read: extract answers from the retrieved documents
- Architecture for Dense Retrieve:
  - Cross Encoder:
    - Passes questions and passages pairs into the model
    - Get the relevance score between pairs
  - Dual Encoder:
    - Encodes the given question into dense representation
    - Compares question representation with the corresponding passage representation

Challenges

- Models are not aggregating sophisticated information into a single dense representation
- There is space to improve retrieval performance with alternative sampling method

Significance of the Project

- Improve the retrieval performance
- Improve the performance of pre-trained model

Background

Previous Work

- Information Retrieval
  - Sparse Representation
    - TF-IDF or BM25: Weight each term by frequency
    - BERT: Weight each term with pre-trained language model
  - Dense Representation:
    - Embedding: Encodes each paragraph and query separately
    - Matching: Measure the relevance of a paragraph to a query
- In-batch Samping
  - AdaBoost: Random in-batch negative sampling
- Pre-training
  - BERT and RoBERTa
  - DPR-PAQ

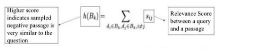
Problem Setup

- Condenser
  - A pre-trained model which establishes structural readiness by aggregating sophisticated information into a single dense representation
- CoCondenser
  - Augment the Condenser MLM loss with a contrastive loss.
- Adaptive Batch Scheduling (ABS)
  - Provides hard negatives instead of random negative sampling during training
  - Maximize the sum of batch hardness scores (the similarity scores between questions and passages over the batch)

Methods

Adaptive Batch Scheduling

- Given: Training instances T = {d1, d2, ..., dt}, where di is the question and passage pair (qi , pi)
- Method:
  - Calculate Hardness Scores:
    - Divide training instances into m batches T = {B1, B2, ..., Bm}.
    - For each batch Bk, define hardness score as the sum of relevance scores between questions and their negative paragraphs over the batch.


Higher score indicates sampled negative passage is very similar to the question
Relevance Score between a query and a passage

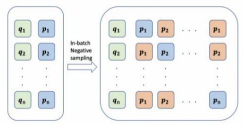
$$h(B_k) = \sum_{d_i \in B_k, d_j \in B_k, i \neq j} s_{ij}$$

  - Schedule Training Instances: maximize the overall hardness scores

Scheduled Training Batches

$$T^B_{scheduled} = \underset{T^B}{\operatorname{argmax}} \sum_{B_k \in T^B} h(B_k)$$

  - Greedy Algorithm:
    - Replace training instances in the initialized sample if this replacement produces a higher hardness score h(B)
  - Beam Search Algorithm
    - Prevent local optimal: Keep track of a list of candidate training instances that might improve h(B)
    - FAISS: Fast indexing and searching for relevant queries and passages
      - retrieves in the current batch



Baseline Pre-trained Language Models

- BERT: Bidirectional transformer pre-trained using masked language modeling
- RoBERTa: Optimized BERT with robust design choices (i.e. dynamic masking, large mini-batches)

Condenser

- Early, Late and Head layers
  - Late layers can only pass information to head layer through CLS
  - Early layers have short circuit to head layers
- Generate CLS representation:
  - Prepend a CLS and embedding
  - Run through the early and late layer backbone
  - Combine late layer CLS and early layers output and run through the head layers.

$$[h^0_{cls}; h^0] = Embed([CLS; x])$$

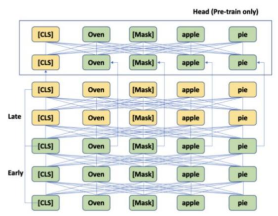
$$[h^{early}_{cls}; h^{early}] = Encoder_{early}([h^0_{cls}; h^0])$$

$$[h^{late}_{cls}; h^{late}] = Encoder_{late}([h^{early}_{cls}; h^{early}])$$

$$[h^{cd}_{cls}; h^{cd}] = Head([h^{late}_{cls}; h^{early}])$$

CoCondenser

- Augmented the Condenser MLM loss with a contrastive loss (unsupervised)
- Loss Function for each batch $\mathcal{L}^{co}_{ij} = -\log \frac{\exp\langle h_{i1}, h_{i2}\rangle}{\sum_{k=1}^{n}\sum_{l=1}^{2} 1_{ij \neq kl} \exp\langle h_{ij}, h_{kl}\rangle}$



Experiments & Results

Data

MS MARCO Passage Ranking (collected from Microsoft Bing search logs)

- Training Set:
  - Randomly select 100,000 queries and its positive passage
- Validation Set:
  - 1,500 queries are selected as the validation set.
    - For each query: 1 one positive passage + 99 passages randomly select from its top 1000 passage

Experiments

- Negative Sampling:
  - Adaptive Batch Scheduling + BM25 similarity
  - Adaptive Batch Scheduling + encoded vector dot-product similarity
  - Adaptive Batch Scheduling + encoded vector dot-product similarity + beam search
  - Comparison: Random and sequential in-batch negative sampling
- Models:
  - Condenser
  - Baseline: BERT and RoBERTa
- Loss Functions:
  - Contrastive loss
  - Learning-to-Rank(LTR)
- Parameters

Training Size	Learning Rate	Batch Size	Epoch
10,000	5e-6	8	15
100,000	5e-6	5	5

Evaluation

- Dev Set
  - Mean reciprocal rank(MRR) $MRR = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{rank_i}$
  - Q: number of queries
  - rank: the position of highest ranked passage in the results
- Full Ranking
  - All 4980 queries and 3,128,597 passages are encoded by the fine-tuned models
  - FAISS is used for retrieving top 100 candidate passages for each query.
  - MRR@100 as evaluation metric

Results

	Dev Set Loss	Dev Set MRR@100	Full-Ranking MRR@100
BERT	4.640	0.422	0.137
RoBERTa	3.491	0.494	0.198
Condenser	**2.376**	**0.592**	**0.253**

Table 1: The best performance of different models using ABS in 10,000 dataset

	Dev Set MRR@100	Full-Ranking MRR@100
Non-ABS-Sequential	0.535	0.227
Non-ABS-Random	0.568	0.233
ABS-BM25	0.578	0.233
ABS-Encode	0.586	0.249
ABS-BeamSearch	0.592	0.253

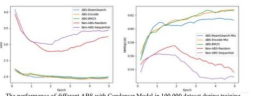
Table 2: The performance of different ABS with Condenser Model in 10,000 dataset

	Dev Set Loss	Dev Set MRR@100	Full-Ranking MRR@100
BERT	2.514	0.503	0.186
RoBERTa	2.256	0.592	0.235
Condenser	**1.946**	**0.627**	**0.281**

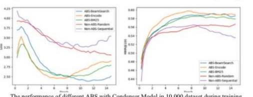
Table 3: The best performance of different models using ABS in 100,000 dataset

	Dev Set MRR@100	Full-Ranking MRR@100
Non-ABS-Sequential	0.561	0.200
Non-ABS-Random	0.577	0.177
ABS-BM25	0.623	0.270
ABS-Encode	0.629	0.281
ABS-BeamSearch	0.611	0.250

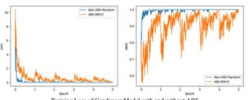
Table 4: The performance of different ABS with Condenser Model in 100,000 dataset



The performance of different ABS with Condenser Model in 100,000 dataset during training



The performance of different ABS with Condenser Model in 10,000 dataset during training



Training Loss of Condenser Model with and without ABS

Analysis

Condenser Model

- Condenser outperforms BERT and RoBERTa with different training strategies and dataset sizes.
- Typical pre-trained LM cannot actively aggregate sentence information into CLS token in the middle transformer blocks
- By forcing late layer to pass information to head layer only through CLS token and short circuiting the information of the early layer, all the transformer blocks can actively aggregate sentence information during training

Adaptive Batch Scheduling

- ABS can improve model performance, especially in large dataset
- ABS finds new hardness negative samples for each query
  - The training loss and MRR@100 increase at the beginning of each epoch, at which time the ABS starts a new scheduling round
  - The training loss of non-ABS training quickly drops near to zero and remains stable
- ABS generates batches always have harder negative samples
  - The training loss of ABS training is always larger than that of the non-ABS training
- ABS finds harder negative samples for each query with larger dataset
  - Consistently and adaptively feeding hard negative samples to models can help better distinguish positive passages from negative passage
- ABS can prevent model from overfitting
  - ABS restarts a new scheduling round at each epoch
  - Each query will have different negative samples at different epochs, preventing the model from overfitting to fixed batched

ABS Similarity Metrics

- ABS based on dot-product similarity has better model performance in the latter training process
- BM25 Similarity
  - Generally find good negative sample
  - Similarity is fixed at start and cannot adaptively change during training
- Dot-product Similarity
  - Adaptively changes during the training process as model continuously learn to provide better encoding
  - Cold start problem
    - It cannot find out real negative samples at the start of training
    - We use BM25 similarity for the first epoch of training

Scheduling Algorithms

- Beam search has better performance in small dataset. Greedy algorithm exceeds beam search in larger dataset
- In theory, beam search should find better negative samples than greedy algorithm
- Due to limited computational complexity, we limit the search range of beam search to top 100 similar queries and passages
- For smaller dataset, this search range is sufficient to find good negative samples
- For larger dataset, beam search creates a bottleneck
- This trade off between computation complexity and accuracy is the key to choose appropriate search range

Conclusion

Main Findings

- Condenser
  - Encode more sentence information into the CLS token
  - Generate better query and passage encoding than BERT and RoBERTa
- Adaptive Batch Scheduling
  - Greatly improve model performance and prevent overfitting
  - Dot-product similarity outperforms BM25 similarity
  - Beam search outperforms greedy algorithm in small dataset
  - Trade off between accuracy and computational complexity

Limitations

- lack of hardware resources

Future Work

- Expand batch size of ABS
- Expand beam search range
- Increase the number of documents retrieved by FAISS