Homework 8

11-967

Due date: 03/23/2025 11:59 PM EST

Language models struggle to perform tasks that require complex reasoning, knowledge that they did not see during training, or interactions with the real world. There are two strategies to improve language model capabilities at these complex tasks: tool-use and retrieval augmentation. Tool-use involves training an LM to call a computer program that runs externally to the LM. Retrieval-augmentation involves conditioning a language model's generations on retrieved information. In this homework, you will train a retriever.

*Note: there are **two** Gradescope submission pages, one for your code and one for your `report.pdf`. Please submit your code to **Assignment** as instructed, and your `report.pdf` to **Assignment (PDF)**.*

Problem 1: Fine-tuning for Dense Retrieval

Dense retrievers, also known as embedding models, use hidden representations to capture the semantic meaning of input sentences, transforming both queries and documents into dense vectors. These vectors enable effective retrieval by measuring the similarity between query and document representations in a continuous embedding space. Unlike traditional retrieval methods, which rely on sparse keyword matching, dense retrieval focuses on semantic similarity, making it better suited for understanding complex relationships between queries and relevant documents.

In this assignment, you will complete the missing functionality in the starter code and fine-tune a small retrieval model. Do not delete this model; you will need it for the next problem.

[Question 1.1] (***Coding, Writing, 5 points***) Transformers produce token-level representations. Pooling is the process of combining those representations into a single sentence-level embedding. In this homework, since we are using a Decoder-only Transformer as the embedding model, you will implement last-token pooling. The starter code provides a dataloader which forces an EOS-token as the last token of both queries and documents. The representation of that token should be used as a sentence-level representation.

DELIVERABLES FOR Q1.1

A. Under `retriever/modeling/encoder.py`, implement `pooling`, which receives the token-level hidden states from the final layer, and returns the pooled representation after applying L2 norm. Hint: By default, the model pads to the right. So you must account for the attention mask when identifying the last token of each sequence in the batch.

B. Encoder-only Transformers (e.g., BERT) often resort to first-token pooling. Explain why it is more sound to use last-token pooling with decoder-only models.

[Question 1.2] (***Coding, 12 points***) Another difference between pure language models and embedding models is the training objective. Ultimately, we want queries and the respective relevant documents closer in the embedding space, while irrelevant documents are further away. This can be achieved through a contrastive loss:

$$\mathcal{L} = -\sum_{i=1}^{Q} \log \frac{\exp(\text{sim}(q_i, p_i^+)/\tau)}{\sum_{j=1}^{Q\times P} \exp(\text{sim}(q_i, p_j)/\tau)} ,$$

where Q is the number of queries in a batch, P is the number of passages per query, p_i^+ is the positive passage for the i^{th} query, $\text{sim}(q_i, p_j)$ is the similarity between the i^{th} query and the j^{th} passage, and τ is the temperature. Note that the starter code uses 1 positive passage and 9 negative passages per query. Hence, the above formulation entails in-batch negatives, i.e., all 10 passages associated with query q_i are used as negative examples for all other queries $q_j, j \neq i$.

DELIVERABLES FOR Q1.2

For the following questions, do not import any more packages other than the ones already imported in the file you are changing. All matrix operations should be conducted with PyTorch operators.

A. Under `retriever/modeling/encoder.py`, implement `compute_similarity`, which receives query and passage embeddings as the input, and returns a query-passage similarity matrix. Use the dot-product as the similarity metric, and apply the temperature parameter.

B. Under `retriever/modeling/encoder.py`, implement `compute_labels`, which should return a list of indexes denoting the position of the positive passages in `p_reps`. Note that `p_reps` contains all the passages in the batch, i.e., P passages per query. The first of every P is the positive for the respective query.

C. Under `retriever/modeling/encoder.py`, implement `compute_loss`, which receives the similarity matrix and the labels as input, and returns the loss. Hint: cross-entropy loss

Your implementations will be graded by passing the respective unit tests.

[Question 1.3] (***Coding, Writing, 6 points***) After implementing the above functions, you should be able to run three scripts. In order:

- `retriever/scripts/train_msmarco.sh` to train your embedding model. (Roughly 60 minutes to run.)
- `retriever/scripts/encode_fiqa.sh` to generate embeddings for the test corpus and queries.
- `retriever/scripts/search_fiqa.sh` to conduct vector search and obtain evaluation results.

The file `retriever/README.md` contains details about each script. You do not need to change any of the provided hyperparameters.

DELIVERABLES FOR Q1.3

A. Present your train loss curve (paste from WandB).

B. Present the Mean Reciprocal Rank achieved by your model on the test dataset. Briefly comment on how to interpret this metric.

[Question 1.4] (***Writing, 5 points***) The starter code implements a retriever through a bi-encoder architecture, i.e., queries and documents are encoded independently. Re-rankers, such as BGE-reranker, use a cross-encoder architecture, where queries and documents are jointly encoded. Refer to this paper for more information on re-rankers.

DELIVERABLES FOR Q1.4

A. Consider the following pseudo-code for the bi-encoder you previously implemented:

```
def bi_encoder(query, passage):
    query_reps = encoder(query)
    passage_reps = encoder(passage)
    similarity = cosine_similarity(query_reps, passage_reps)
    return similarity
```

Assuming you have access to a linear layer `Linear` $\in \Re^{\text{hidden\_dim} \times 1}$, present pseudo-code for a cross-encoder.

B. Name one advantage and one disadvantage of the bi-encoder architecture when compared to cross-encoders.

C. Given a large document collection (millions) and access to both a bi-encoder and a cross-encoder, describe a strategy to use both models effectively in an information retrieval system. Explain how you would balance efficiency and effectiveness in this scenario.