Homework 3 (Revision 0)

11-667 Fall 2024

Due date: 14:00 October 17, 2024

As we have learned in class, language models struggle to perform tasks that require complex reasoning, knowledge that they did not see during training, or interactions with the real world. In class, we learned about 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 very simple tool-use LM, where the tool is a basic calculator. You will also train a retriever and build a system for retrieval-augmented generation.

Note: Unlike for HW1, there are ***two*** *Gradescope submission pages for HW1, one for your code and one for your* `report.pdf`*. Only* `report.pdf` *files submitted to the "written" Gradscope submission page will be graded.*

Problem 1: Language Model with a Calculator Tool

Why do we want language models to use tools anyway? It turns out that many tasks are difficult, or fundamentally impossible for a language model to perform. One such task is basic arithmetic. In this question, you will explore how to teach a language model to use calculators in math word problems.

[Question 1.1] (***Writing, 5 points***) First, let's convince ourselves that numerical computation is indeed difficult for even the best language models. Go to OpenAI's API playground (https://platform.openai.com/playground/chat?models=gpt-4o) to experiment with inference on GPT-4o. [1] Note, we are using the playground interface because the "ChatGPT" interface already has tool-use built in. Try to come up with an instruction that:

- requires an arithmetic expression involving basic operations (e.g., +, -, *, /) to answer
- the output of GPT-4o is off from the expected value by $> 10\%$.

Feel free to use any sampling strategy that you like.

DELIVERABLES FOR Q1.1

In your report, write down the following five items:

- prompt
- model output
- result expected
- relative error rate
- choice of model (if choosing one other than GPT-4o)

[Question 1.2] (***Coding, 5 points***) We are going to use an elementary math word problem dataset, ASDiv for training our language model to use a calculator. To do this, we have pre-processed the ASDiv dataset for you into a format that allows language models to learn *when* and *how* to invoke the calculator tool. Inspect the dataset at https://huggingface.co/datasets/yimingzhang/asdiv, and examine the text and target outputs. The model will be fine-tuned to produce `target` when prompted with `text`. For

[1] You will need to set up billing information with OpenAI in order to use the playground. If this is not an option for you, you may alternatively use any other frontier language model which does not have tool use incorporated (e.g. LLaMA Instruct).

example, given the prompt ‘‘`Question: Rachel bought 8 music albums online. If each album had 2 songs, how many songs did she buy total? Answer:`’’, the model is trained to first parse the expression to be calculated inside angular brackets (‘‘`«8*2»`’’), followed by the value of the the expression (‘‘`16`’’).

During training, the model is just fine-tuned to produce the entire target (‘‘`«8*2»16`’’). You will implement an inference-time check `can_use_calculator()` that returns `true` if the angular brackets are completed (i.e., generation ends with ‘‘`»`’’), which suggests that the calculator tool can be used.

DELIVERABLES FOR Q1.2

Implement `can_use_calculator()`.

[Question 1.3] (***Coding, Writing, 8 points***) Implement `use_calculator()`, the function that calls a calculator (`safe_eval`) on partial model generation and appends calculator output back to the input. For example, on the input string ‘‘`Question: Rachel bought 8 music albums ... Answer:«8*2»`’’, `use_calculator()` should return ‘‘`Question: Rachel bought 8 music albums ... Answer:«8*2»16`’’. Return the input string if it does not end with a well-formed arithmetic expression.

DELIVERABLES FOR Q1.3

A. Implement `_use_calculator()`.

B. What if we were to use the built-in Python `eval()` function instead of `safe_eval()` for numerical evaluation? Specifically, what can go wrong when executing unsanitized language model output? Comment with at most 3 sentences.

[Question 1.4] (***Writing, 5 points***) Now you should have all you need to fine-tune a small language model (Pythia-1b) that can use a calculator. Run `python src/calculator/main.py` to train and evaluate the model (this should take no more than 10 minutes). You should now expect significant gains in accuracy when the model is provided calculator access.

DELIVERABLES FOR Q1.4

Report test accuracy on ASDiv, with and without calculator access.

[Question 1.5] (***Writing, 6 points***) Model generations for the test instances (both with and without calculator access) are dumped to `pythia-1b-asdiv/eval.jsonl`. Use this file to answer the following questions.

DELIVERABLES FOR Q1.5

A. Find a test instance where the fine-tuned model produces an incorrect answer without calculator access, and produces the correct answer with calculator access. Report the example prompt, as well as generations with and without calculator access. Include an one sentence discussion.

B. Find a test instance where the fine-tuned model produces an incorrect answer even with calculator access. Where did things go wrong? Report the example prompt, as well as generations with and without calculator access. Include an one sentence discussion.

Problem 2: 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 2.1] (***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 Q2.1

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 2.2] (***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 Q2.2

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 2.3] (***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 Q2.3

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 2.4] (***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 Q2.4

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.

Problem 3: Building a RAG System

Retrieval augmentation is a useful technique to improve LLMs' performance in answering factoid queries, especially those regarding niche domains. In this problem, you will use the retrieval model you trained in the previous section to build a small RAG system.

[Question 3.1] (***Writing, Coding, 5 points***) You will be using questions from TOFU, and an instruction-tuned Pythia 6.9B model.

DELIVERABLES FOR Q3.1

A. Do you expect the chosen language model to be able to answer questions from the TOFU dataset? Why?

B. Run `.retriever/scripts/generate_without_rag.sh` and present the questions and respective completions.

[Question 3.2] (***Writing, Coding, 5 points***) The code under `.retriever/driver/rag.py` is built from the inference code for Pythia on Homework 2. You need to extend its functionality by implementing a function that augments prefixes with relevant passages.

DELIVERABLES FOR Q3.2

A. Implement the missing functions, and run `.retriever/scripts/generate_with_rag.sh`. Present the queries and respective completions.

B. Look for the answers in the TOFU dataset, and comment on the effectiveness of the approach.

[Question 3.3] (***Coding, Writing, 5 points***) The performance of RAG systems is conditioned on the quality of the retrieval step. Look at the top-N passages retrieved for each query. In this case, you should see the relevant passage very close to the top.

DELIVERABLES FOR Q3.3

A. In more difficult scenarios where the retriever won't be as effective, using a re-ranker is a common step of the pipeline. Besides the cross-encoders introduced in the previous exercise, LLMs as list-wise re-rankers have also been studied. Refer to this paper for more details. Name one advantage and one disadvantage of cross-encoders when compared to LLM list-wise re-rankers.

B. In Question 3.2, the prefix was augmented with the top-1 passage. Present and comment on changes in the generation results if we augment the questions:

- With the top-10 passages in-order.
- With the top-10 passages shuffled.

Problem 4: Use of Generative AI

If you used Generative AI for any part of your homework, you should fill out this question. Failure to do so is an academic offense and will result in a failing grade on the homework. You may omit this question from your submission if you did not use Generative AI.

[Question 4.1] (***Writing, 0 points***) Did you use a coding assistant (e.g. GitHub Copilot) that was built into your code editor? If yes, which one did you use? Describe what parts of the code you wrote yourself and which parts it wrote for you.

[Question 4.2] (***Writing, 0 points***) Did you converse with a chatbot agent (e.g. Gemini, Claude, or ChatGPT) to help with either the coding or conceptual questions on this homework? If yes, include a table that contains the following information:

- The prompt you passed to the agent.
- The agent's output for that prompt.
- A brief description (∼1 sentence) of which homework question(s) you used this output for, and how it was incorporated into your final answer.

[Question 4.3] (***Writing, 0 points***) Did you have any other use of Generative AI that you would like to disclose?