Report

1. Introduction

1.1 Task Overview

This task is going to dive into LLM and utilize the most likely ability of LLM, by find the correct answer in the multi-choice question without any fine-tuning or retraining process.

1.2 Background and Related Work

There are two kinds of QA task, one is to deduce the answer from answer, another is to choose the word that is initially in the question and extract this word as answer. Before Vanilla Transformers, people used RNN to do text classification task to implement multi-choice question, which means that practitioner needs to train lots of models for different domain. After the advent of Vanilla Transformers and BERT, the paradigm of "fine-tuning" was coming up, which only asks people prepare a small piece of dataset from specific domain and fine-tune the pre-train model on these data. It saves lots of time to train a model from scratch and the performance of "fine-tunning BERT" is SOTA. Since the number of model parameters is becoming bigger and bigger, more and more people couldn't afford money to train or fine-tune an LLM. So this phenomenon boosts the "prompt tuning" paradigm. Only give the LLM the prompt to inference the result.

2. Methods

2.1 baseline

the whole workflow could be considered as the following step:

1. feed the whole training dataset via embedding model get a 2D tensor.
2. feed every validation dataset via embedding model and calculate the inner product with the tensor from step 1(the more inner product it is, the more closed/similar these two tensors have) as the score, sort the score and choose the most likely N sample from training dataset.
3. build the prompt by using filtered dataset from step 2, in 2 styles, and use LLM to inference the result.
    - Version 1: "Question: xxx \nCandidate Answer: xxx\nGold Answer:"
    - Version 2: "Question:xxx\nAnswer:"

2.2 improvement

What I improve:

1. Change the function of generating prompt. It only receives a parameter of max_len and tries to fill the prompt to max_len-100 as closed as it can, not just specify the number of

question-answer pairs. The reason why I leave 100 token is to make sure answer must have enough space to write down Since the answer won't be too long, so no useful information will be truncated during encoding or decoding.

2. Use reverse parameter to format the prompt. If reverse is true, then the max similar they are, question-answer pair appear first, and vice versa.

3. Find the gap regarding performance of phi1.5 and phi2.

4. Compare the performance of embedding models between bge-little, bge large, bge-m3, llm-encoder. Different embedding model might derive different output that affects the similar metrics.

5. Linear interpolation to implement length extrapolation. (It requires much more memory, like 32GB to run it. What's more, it needs to fine-tune the model to adjust the new positional encoding, which is really time-consuming and money-consuming. GIVE UP).
After thinking it carefully, I won't use linear interpolation or NTK methods. What I implement is to utilize the sliding window, cut the long text into more than one chunk and use decay weights to calculate the weighted score. (Like the idea of XLNet and transformerXL)

6. Hyperparameters tuning

7. Fix some bugs

3. Experiment

3.1 Origin model:

the performance of origin model is as follows: (eval_few_shot.py)

easy + max_len 1024 + prompt 2 + N 8 + reversed False	**0.80175**
easy + max_len 1024 + prompt 1 + N 8 + reversed False	0.62982
easy + max_len 1024 + prompt 2 + N 8 + reversed True	0.78771
challenge + max_len 1024 + prompt 2 + N 8 + reversed False	**0.5217**
challenge + max_len 1024 + prompt 1 +N 8 +reversed False	0.4414
challenge + max_len 512 + prompt 1 + N 8 + reversed False	0.2140

After doing experiments, I found that the following conclusion:

- the more related training examples in the prompt (max_len is bigger), the more performance it gets.
- prompt 2 has better performance than prompt 1 has in the baseline.

- reversed True affects LLM inferencing process, LLM might be likely to process the most similar text first.

Firstly, find the reason why the baseline doesn't work well, talk about what baseline might miss and then take action to improve.

- the hyperparameters N and max_len might collide with each other, which means that if N is bigger and max_len is small, tokenizer will truncate those last characters. Based on the prompt you defined, the model won't know what you want it to inference.
- Does Phi1.5 really have the capability to inference?
    - Understand that question might help us figure out why prompt 2 has better performance in Phi1.5 than prompt 1 has. Since prompt 2 doesn't have candidate answer information but it has much better performance in phi1.5, it's normal that prompt 1 should be better than prompt 2, but the real situation is not true.
    - Just execute **tokenizer.batch_decode(output)** to print what Phi1.5 inference and find that the output is full of "computer language code" and Phi1.5 just wants to write code under any other prompts, which is really weird, because the prompt doesn't contain any code.
    - So I can conclude that phi1.5 might don't have enough ability to handle this task. I changed phi1.5 to phi2 and check the output after feeding the prompt, the output is quite reasonable, although it contains some repeated text.
- Based on the specified hyper-parameters set, do I choose the most similar come first or the less similar come first in the prompt?
    - After experimenting, it's better to show the most related QA pair first, which might teach LLM to learn at the right direction.

3.2 Improvement

3.2.1 Improvement 1 ---- modify prompt format

The improvement I did has been shown in the part 2.2. I will illustrate the performance of experiment directly.

Still use Phi1.5 and only change the way of building prompts like mentioned. After filling as many similar training data pairs as it can to the prompt, the performance is better in the easy dataset and worse in the challenge dataset, compared with the baseline.

easy + max_len 1024 + modified prompt 2 + reversed + phi1.5	0.7912
easy + max_len 1024 + modified prompt 1 + reversed + phi1.5	0.6070
easy + max_len 2048 + modified prompt 2 + reversed + phi1.5	**0.8140**
easy + max_len 1024 + modified prompt 2 + not reversed + phi1.5	0.7894
challenge + max_len 2048 + prompt 2 + revered + phi1.5	0.4983

3.2.2 Improvement 2 ---- choose better prompt and change the model

After executing tokenizer.batch_decode(output), the phi1.5 inferences some messy code in the outputs, which is ridiculous, because there doesn't have any code in the prompt. It's time to change phi1.5 to phi2. Firstly, test the performance of phi2 on the baseline, just change the model.(eval_fewshot.py)

easy + max_len 1024 + prompt 2 + N 8 + not reversed + phi2	**0.8438**
easy + max_len 1024 + prompt 2 + N 8 + not reversed + phi1.5	0.80175
challenge + max_len 1024 + prompt 2 + N 8 + not reversed + phi2	**0.5752**
challenge + max_len 1024 + prompt 2 + N 8 + not reversed + phi1.5	0.5217

The performance has been increased significantly after using phi2, compared with phi1.5. So the following experiments will be under phi2 + modified prompts. (improvement_few_shot.py)

easy + bge-small-v1.5 + modified prompt 2 + max_len 1024 + phi2	0.8473
easy + bge-small-v1.5 + modified prompt 2 + max_len 2048 + phi2	OOM (16GB)
challenge + bge-small-v1.5 + modified prompt 2 + max_len 1024 + phi2	0.5986
challenge + bge-small-v1.5 + modified prompt 1 + max_len 1024 + phi2	**0.7625**
easy + bge-small-v1.5 + modified prompt 1 + max_len 1024 + phi2	**0.8842**

Since prompt 1 has more information about candidate answer, it should be better to inference the answer. However, under phi1.5 model, the performance of prompt 1 is much lower which means that Phi1.5 model doesn't have enough ability to tackle with this problem. After changing the model to Phi2, the performance of prompt 1 is much better than the performance of prompt 2.

3.2.3 Improvement 3 ---- choose different embedding models

Because the previous part has discussed that the reason why the most likely data pair should come first, the following part will focus on how well bge-small model can could perform, compared with bge-large model, newly releasing bge-m3 model, since these embedding models could derive different embedding vectors and affects the similarity metric.

easy + bge-small-v1.5 + modified prompt 1 + max_len 1024 + phi2	0.8842
easy + bge-large + modified prompt 1 + max_len 1024 + phi2	**0.8947**
easy + bge-m3 + modified prompt 1 + max_len 1024 + phi2	0.8859
challenge + bge-small-v1.5 + modified prompt 1 + max_len 1024 + phi2	0.7625
challenge + bge-large + modified prompt 1 + max_len 1024 + phi2	0.7625
challenge + bge-m3 + modified prompt 1 + max_len 1024 + phi2	**0.7692**

After checking these comparisons, we could find that bge-small has the slightly lower performance than bge-large or newly released bge-m3 on this task.

3.2.4 Improvement 4 ---- length extrapolation

There are many ways to implement length extrapolation by using the property of RoPE, such as linear interpolation, NTK, etc. But I don't think I need to follow that way because in this task there have some following flaws:

1. How to communicate with different chunks because every chunk must end with the origin question, origin question in the different chunk has the different positional embedding should not be a good idea. What's more, which chunk should be responsible for the output scores are also needed to think about.
2. It needs to be fine-tuned on some long text QA datasets under very limited training data.
3. Very time-consuming and money-consuming.

What I did is to divide the long text that has been sorted based on similarity, into more than one chunk which ends with the origin sentence, use the sliding window to calculate the score one by one, and use weight decay method to get the final score as prediction.
The reason why I use weight decay is that the previous QA pair has the high similarity, the score needs to attend more to the more related chunk. The weight decay for every chunk I set is 0.5 (I just try that value, which doesn't mean that it works best, because it costs lots of time to find a satisfactory one, this part is only show whether length extrapolation works or not)

Just compare the performance with what I did:

easy + bge-large + modified prompt 1 + max_len 1024 + phi2	**0.8947**
challenge + bge-m3 + modified prompt 1 + max_len 1024 + phi2	0.7692
challenge + bge-large + modified prompt 1 + max_len 1024 + phi2 + num 2	0.7725
easy + bge-m3 + modified prompt 1 + max_len 1024 + phi2 + num 1	0.8877
easy + bge-m3 + modified prompt 1 + max_len 1024 + phi2 + num 2	0.8912
challenge + bge-m3 + modified prompt 1 + max_len 1024 + phi2 + num 1	0.7725
easy + bge-large + mofied prompt 1 + max_len 1024 + phi2 + num 2	0.8929
easy + bge-large + mofied prompt 1 + max_len 1024 + phi2 + num 1	**0.8947**
challenge + bge-large + modified prompt 1 + max_len 1024 + phi2 + num 1	**0.7792**

It's not hard to find that the normal model without length extrapolation has been already good enough in the easy question, so even if the model with much bigger context window doesn't have a better performance in the easy QA pairs. But the model with bigger context window has better performance in the challenge QA pairs, which means that the phi2 model needs more text to deduce the answer well.
I think LLM with much bigger context can utilize the ability of inducing in the challenge problem. There are not too many experiments to find out the best or more satisfactory parameters set. Which hyper-parameters set I have used is randomly chosen, so I think there has more space to improve.