Final Project (100 pts)
Finetuning a Domain-Specific Chatbot

Instructor: Jorge Mendez-Mendez

Instructor meeting due: Friday, November 22nd, 2024
Final report due: Friday, December 6th, 2024 at 11:59 pm
Group size: Minimum 1, maximum 3 students per project

For our final course project, we are going to explore today's most popular use of deep learning: large language models (LLMs). The final outcome of the project will be a language model specialized in knowledge of some domain of interest to you. I will use the contents of ESE 577 as a running example in this write-up, but you are welcome to use data from any domain of interest to you, for which you have data that is not publicly available online. We have by now learned two of the key components of specializing an existing LLM to our own domain: transformer architectures and finetuning. The typical pipeline for adapting a general-purpose language model to a specialized domain (such as ESE 577) is: download the weights of an open-source language model, create a dataset from your target domain, and finetune the pre-trained language model on your dataset.

However, without proper care, the finetuning step would require computational resources that far exceed those found in consumer hardware (or in the free tier of Google Colab). The challenge is that high-quality LLMs typically have several billion parameters—for the Mistral-7B model that we will use in this project, using full-precision floats, the weights alone would require 28 GB of GPU memory; backpropagation would consume another 28 GB; and we still must account for our data batches. To get around these issues, we will explore parameter-efficient finetuning (PEFT), using low-rank adaptation (LoRA) and quantization.

Even though these techniques will make finetuning feasible on the free tier of Google Colab, a single run of finetuning might take multiple hours. I strongly encourage you to start early.

1 Mistral-7B

As a base model, we will use Mistral-7B, a transformer model with 7 billion parameters, whose weights are open source. There are three officially supported versions:

- Mistral-7B-v0.1: the base model, pretrained on next-token prediction, like the GPT we studied in class. This model is primarily useful for text completion.
- Mistral-7B-Instruct-v0.1: an adapted version of the base model, finetuned to act as a chatbot on a dataset of input/output pairs.
- Mistral-7B-Instruct-v0.2: an improved version of the finetuned chat-oriented model.

You are free to use whichever version of the model you prefer, but keep in mind that "Instruct" models will lose their ability to act as chat models if you finetune them for text completion, and conversely the base model will fail to act as a chatbot if you finetune it on the relatively small amount of data you will use. As such, your choice of model should be coupled with your choice of data processing, described below.

2 Data processing

Using text-completion data requires the least amount of effort in data processing: you can simply use any text you have available from your domain of interest (e.g., ESE 577) and train your model to predict the next token given the previous sequence of tokens. If you follow this route, you should likely use the base Mistral-7B-v0.1 model, since you will lose any chatbot abilities during finetuning anyway. Your training setup should be: given a sequence of tokens, minimize the NLL loss on the next token.

Alternatively, you could convert our ESE 577 into a dataset of question-answer pairs. There are (at least) two possibilities here:

- Write a set of approximately 100 question-answer pairs manually, and use that as your training data.
- Use a (more) powerful LLM to generate question-answer pairs for you, and use the synthetic data for training. For this, you could use Google Gemini Flash 1.5, which is free and quite powerful. Here's a rough list of steps:
  - Process the data so that you end up with a single list of paragraphs. If you are including content from slides, which have little text, you might want to merge bullet points from one slide into a single paragraph, so that there is sufficient content to generate a question-answer pair.
  - Write a prompt that encourages the Gemini to create one insightful question-answer pair from the paragraph, such that answering the question with the given answer demonstrates knowledge of the content of the paragraph. In your prompt, request that the format of the output be:

    `<s>[INS]@ESE577.<question>[/INS]<answer></s>`
  - Loop through the paragraphs you created in step 1 and prompt the system to generate a question for the given paragraph.
  - If you include quizzes, you can directly turn them into questions by processing them in the required format.

The synthetic approach can easily generate more data, but real data is often beneficial for training chatbots. Of course, you can also use a combination of the two.

Out of the three options outlined here (text-completion, manual question-answer pairs, or synthetic question-answer pairs), you are free to choose whichever you prefer. In your final report, you should clearly state your choice and outline its pros and cons.

As your "base" data (which will directly be your dataset in the text-completion case, and will be used to generate question-answer pairs in the chatbot case), you can any source of text for which you have non-public data. For example, for ESE 577, you could use text from slides, quizzes, homeworks, MIT course notes, Piazza, or your own notes. You are welcome to complement this data with text from external references you have read throughout the course (e.g., blog posts, book chapters, video transcripts), but make sure that you also use your own private data.

For any of these approaches, you should also develop a validation dataset.

For text-completion data, it is sufficient to split the original text 70-30, though you should be careful how you split your data. For example, if you train on the first 8 weeks of content and test on the final 4 weeks, your model will not know the content of the final 4 weeks. Or if you train on "`In linear regression, our hypotheses take the form of a <blank>`" and evaluate on "`In linear regression, our hypotheses take the form of a linear <blank>`" your validation data overlaps too much with the training data. A good split should take some paragraphs for training and other paragraphs for testing.

For question-answer data, it is probably a good idea for your validation data to be multiple-choice questions (instead of broader open-ended questions). This makes it easier to measure accuracy directly.

3 Low-rank adaptation (LoRA)

LoRA is a popular technique for finetuning large models efficiently. The idea is that, given a set of pretrained weights $W_{\text{pretrained}}$ for a layer, we can create a finetuned set of weights as:

$$W_{\text{finetuned}} = W_{\text{pretrained}} + W_{\text{left}} \cdot W_{\text{right}} \ .$$

The shapes of the matrices are:

- $W_{\text{pretrained}} \longrightarrow d \times d$
- $W_{\text{left}} \longrightarrow d \times d'$, with $d' \lll d$
- $W_{\text{right}} \longrightarrow d' \times d$
- $W_{\text{finetuned}} \longrightarrow d \times d$

The original weights $W_{\text{pretrained}}$ are left in their original values (meaning that we do not compute any gradients on $W_{\text{pretrained}}$; we just treat these as constant), and we only train the two matrices W_{left} and W_{right}, which reduces the number of trainable parameters from d^2 to $2dd'$.

4 Quantization

Roughly, the idea of quantization is to reduce the floating-point precision of the learned weights and data, e.g., from 32-bit full precision to 8-bit integers. In practice, modern methods go as low as 4-bit precision. We won't go into the details of how this is achieved, but we will use these techniques from existing libraries.

5 Finetuning on Google Colab

We will not implement our own versions of LoRA or quantization. Instead, we will leverage existing libraries that already implement these tools. You will write code for loading your own data, looping through your data for training (over, say, 100 epochs), and evaluating on your validation data.

You will also write code to interact with the trained model.

6 Numerical evaluation on validation data

While the real test of a language model's performance is human evaluation (which we will go over in the next section), it is still useful to attach numbers to our model's performance. For this, you will implement evaluation code that measures an appropriate measure of performance for your data.

For example, for text-completion data, you can directly measure the NLL loss on validation data. Alternatively, you could measure the *perplexity* of your model on validation data. For question-answer data, we can simply measure the model's predictive accuracy on multiple-choice questions. Note that this requires parsing the models text output into a predicted class (e.g., map the model's "a" output to class 0).

It is useful to periodically run evaluation on the validation data during training to check if your model's performance improves over time. Your first evaluation should be *before* any training, to have a baseline of performance from the base model, and then you should periodically evaluate after a number of epochs of training.

7 Human evaluation

Use the provided sample code to interact with your trained model. How do its outputs look? Are you satisfied with performance? If not, how could you improve it? Include sample interactions with your trained model in the final report!

8 Getting help

This project is a step up from the homeworks you have done so far. While you should have the foundational knowledge required to implement each of the steps above, it could be challenging to get a complete end-to-end pipeline working for the first time. (Though, at the outcome, you will have a model that is on-par of today's most powerful local models!)

During this month-long project, you should leverage office hours and Piazza to ask questions. You are also free (and encouraged) to look around online for other people's solutions to similar problems. **The one**

caveat is that any code you turn in should be entirely written by yourself—you are not allowed to copy text found online.

9 Deliverables

There will be two components to your project grade, described below: a meeting with me, and a final report.

9.1 Instructor meeting (40 pts)

I will hold a midpoint meeting with each project group. Meetings will typically happen during office hours, but if you have a conflict during office hours, it is your responsibility to reach out through Piazza to schedule a separate time.

During our midpoint meeting, I will expect you to show significant progress in at least 2 of the following:

- Data processing – have you already transformed ESE 577 text into a suitable format for finetuning Mistral-7B?
- Finetuning code – have you written code for looping through your data to finetune the model's weights?
- Running a sample training loop – have you run at least a small (e.g., over 1 or 2 data points) training loop using LoRA and quantization?
- Numerical evaluation – have you implemented the numerical evaluation code and integrated it into the training loop?
- Human evaluation – have you interacted with the base or finetuned model through the human evaluation code? What where your findings?

9.2 Final report (60 pts)

The final report should have a length of 4–8 pages (longer doesn't mean better!). It should contain the following components:

- Code: You should at a minimum submit your Google Colab notebook used for finetuning the data. If you used a separate script to process the data or in any other part of your project (e.g., to create plots), you should submit that as well.
- Abstract: briefly summarize what you did (including your main choices, e.g. text completion vs chatbot) and what your outcomes were.
- Introduction: briefly motivate the finetuning approach, the use of LoRA and quantization, and summarize/motivate any additional choices that you made.
- Method: go into details of your training method. How did you process your data? How many data points did you create? How long did you train for? Which hyperparameters did you choose and why? (Hint: full-blown hyperparameter tuning may be difficult for this scale of project; it may be more appropriate to browse online for typical choices of d', the number of epochs, and the learning rate.) What loss function did you use? What was your numerical evaluation metric?
- Results: show me how your approach worked! You can include a "learning curve" that shows how the numerical performance on validation data varied during the training loops, or a pair of numbers for initial (base) and final performance. You can also include any results of prior attempts that you made that worked better/worse than your final approach. Include screenshots or copy-pasted text interactions with your final model on ESE 577 content. Be creative!
- References: add any citations that are relevant here, including blog posts, videos, books, or research papers that you read/browsed/skimmed for this project.

You should submit a single `.zip` file to Gradescope, including a folder with all your code and a single PDF file of your report. Use this LaTeXtemplate to write your report.