Course Project

Proposal and Final Submission Instructions

Project Proposals Due: October **28** by 11:59 pm
Final Projects Due: November **21** by 11:59 pm (Friday after our 13th class).
Note that the following Wednesday is an EP holiday, and the last class is Dec 3.

For the remainder of the course your programming homework will be to complete a final project rather than weekly programming. There will still be occasional readings and short answers- look for these separately.

The final project will consist of implementing a variation or alternative to the GPT transformer architecture, tokenizer, training loop, or inference loop, and testing how this impacts the model. You may invent your own variation of interest or implement something from existing literature. You may alternatively build an application on top of an LLM, but will still need to find ways to profile it quantitatively and lead to technical insights.

You should test your variation through metrics that show how your variation compares to the model we implemented together in previous homeworks. How do your changes impact training time, convergence, performance (or whatever the relevant factors may be)? The results of these experiments will be written into a short research report that describes your project and findings.

To assist in creating compelling project ideas, you will first submit a project proposal. You will be provided feedback on your individual project ideas and expectations for a successful final submission. Example ideas are included at the end of this document. You can select one of these if you wish.

<u>Note</u>: You do <u>not</u> need to *improve* the existing GPT architecture, but simply want to focus on a variation that is well-motivated. A null result is fine, as long as you can explain why current practice is preferred over your idea.

The final project will have the following components, which are each graded separately:

1. Project Proposal (10%): A written proposal including your intended variation of the GPT model, experiments to evaluate, and hypothesis as to the impact of the variation. Include all team member's names. The proposal must show critical thinking as to the variation implemented, why it may be significant, and how it will be evaluated.

The proposal is also an opportunity to get feedback on your ideas. You will be expected to incorporate proposal feedback into your final submission.

2. Programming Component (40%): A working version of your variation and code to use your implementation, with instructions to run. This will be graded similarly to other programming assignments (clarity, functionality, ease of use, etc). Your results should be reproducible by running your provided code.

3. Final Report (50%): A multi-page report detailing the relevant aspects of GPT models, your proposed change and its motivation, a comparison between your change and the version of GPT from homeworks, and conclusions that can be drawn from these results. This will follow a typical research paper / lab report structure as you may have seen in previous courses. See below for more details.

Proposal (Due October 28)

In a few paragraphs, please explain the following:

- Your project idea (What you intend to implement)
- How you expect your implemented component to differ from the GPT version from our homeworks (or if you are unsure, explain why this idea is of interest)
- Specific experiments or comparisons you intend to make. What are the key metrics that you are interested in?
- If applicable, any datasets or libraries you will use.

As an example, if your project was to implement binary BPE (instead of character-level BPE from homeworks), you may want to touch on things like:

- Expected utility of this change- this would allow support for many other languages and symbols
- Dataset- perhaps you try training on a corpus of text that is in a non-alphabetic language
- Comparisons- how would this compare to simply using <unk> for unsupported characters? How does the new tokenization impact things like vocabulary, training time, performance?

Project Programming (Due November 21)

An implementation of your project in code, which has key routines that are easy to run and well-documented. How you present this is up to you, but it should be packaged well for users besides yourself. Some typical items in this regard might include:

README.md (or README.txt) - a README file that explains your code and how it is intended to be used. Explains outputs of the code, the order things are run, etc. If your code is configurable, explains how to configure it for various experiments.

config.json (or similar) - a file that holds key parameters for a script. This is common practice once the number of settings/hyperparameters becomes hard to track, i.e. for a training routine.

main.py - a file to run your key experiment(s), such as a training loop or tokenization routine.

installation.txt - Instructions on how to install any libraries or access necessary datasets. You could also include this type of information in a README.

As with homework programming assignments, this should be well commented, formatted, and readable. It will be graded to a higher standard than typical homework.

Project Report (Due November 21)

Please compile a paper / lab report on your project that explains the motivation behind your project, what you implemented, and how you found it to change the behavior or performance of a GPT model. You should include diagrams, tables, and/or plots to convey results. Aim for around 4 or 5 pages total. The specific formatting is up to you, but it should be presentable and professional (word doc, pdf of latex or markdown, etc. Do not submit a plain text file).

It is recommended to include the following types of sections in your document, as you might find in a research publication:

1) Introduction: Summarize your project. What did you implement and why? What was the expected outcome of your changes to the GPT model, or what functionality were you trying to achieve? What were your key results and conclusions?

2) Background: Explain the relevant mechanisms of the GPT model in your own words, and how your implementation is intended to differ. If someone knew about LLMs at a cursory level (and had general ML knowledge), what would they need to understand about your project? What are some relevant works or sources you consulted (if applicable.) You do not need to have a formal "related works" section. Just give the reader an idea of what we are talking about here.

3) Methods: Explain in detail how your project works, any datasets or libraries used, and specific experiments you performed or metrics you used. In short- what did you make and how does it work?

4) Results: After implementing everything above and running it, what did you find? Include any numbers, tables, plots, etc. If you had qualitative comparisons (i.e. inspected the generation of two models side by side), include some examples.

5) Conclusions: What did you learn from this project? What can we conclude from your results? You can also take this opportunity to mention any obstacles that you faced or surprises along the way- did something work exceptionally well (or not)?

6) Impact and Future Work: What would you do if you had more time to continue the project? How might your results impact large-scale models that may be trained on 1000s of GPUs and/or released to the masses? If you had access to those types of resources, what would you want to try next?

Example Project Ideas

You may choose something from this list to implement if you wish, but you are also free to come up with your own ideas in a similar vein. These are organized by our previous modules:

- Tokenizers:
  - Byte-level BPE
  - Tokenization scheme for special domains like math, code, etc.
- Embeddings:
  - Rotary Position Embeddings
  - Word2Vec or similar technique, used in conjunction with LLMs in some way
  - Your own learned embedding scheme
- Attention:
  - Sliding window attention / longformer
  - One of the many other transformer variants
  - Cross attention
  - MLA
- Transformer Model:
  - Exploration of model architecture settings/sizes
  - Your own variant of the transformer block (would need to be well motivated)
  - ViT, DiT, or other transformer-based model for other applications
- Training:
  - Exploration of training hyperparameters
  - Exploring training curricula (presenting data in a non-random order)
  - Exploring / defining a new scaling law for some aspect of the GPT model
  - Training in a completely different domain (i.e. code generation).
- Inference:
  - Inference with KV-Caching
  - Custom sampling strategies
  - Using output statistics to detect AI-generated text
  - Watermarking (hiding signatures in the text via sampling so that you can determine that your model was the source of the text)
- Later Lectures or Other:
  - Fine-tuning for a specific downstream task
  - Implementing LoRA or similar PEFT method
  - Enabling Chain-of-thought prompting
  - Setting up a custom RAG system
  - Parallelized training or inference
  - Instruction tuning (could overlap with first two bullets in this section)
  - Training a reward model

Note that some of these require more compute than others. Do not pitch a project that you cannot reasonably complete! For example, if you pitch an exploration of training hyperparameters, that means you will need to train at least a handful of models.