Introduction

This course provided a well-rounded general view of the current transformer-based generative language models, their use, evaluation and operating principles. We got a lot of very well written notebook exercises that enable us to get a basic understanding of some central workflows. These also allow us to expand on them to build our own projects. I attended this course mainly because I was interested in RAG and to my great excitement, Dima had prepared a very comprehensive notebook environment for the purpose of teaching that. Unfortunately I had to prioritise other work at this particular time, therefore my report will be not as comprehensive, but I still appreciate the effort put into this course and it would be good to see it continue.

What are tokenizers?

Tokenizers are a collection of encoding or compression software that split symbol sequences into sub-parts and cast them into a numerical form. They generally build a dictionary that maps tokens in the vocabulary into an id. These tokens are then fed into the LLM to train on.

Why are they important for language modelling and LLMs?

During training, LLMs do computational operations on tokens and they have to be able to calculate relationships between tokens as sub-parts of language. This requires some method of splitting the original text (or other) data into parts, such that the system can calculate the relations of the sub-parts.

In the case of generative AI, you want the system to take your input and give you the most probable sequence of symbols conditioned by your input and based on its training data and additional guardrails. Let’s say you don’t do that and only input the documents as a single object. Then the system would work more like a very rough and weird document retrieval system, where it generates the most likely

documents based on your input. The resolution of the tokenization affects the function of the model.

What different tokenization algorithms there are and which ones are the most popular ones and why?

Subword tokenizers are a family of tokenization algorithms that split words into smaller sequences. They are probably the most common tokenization algorithm currently being used.

Bojanowski et al from Facebook AI research found in 2017 that sub-word character n-grams of 3-6 characters exhibited the fastest training time among candidates. This length roughly equals a typical morpheme in english. Sub-word tokenizing lowers the dimensions of the vocabulary, as sub-words can be used to calculate many words in the same way that the alphabet consists of only 26 letters but can be used to compose the entire vocabulary. This makes training the model quicker. also, it can help encode out-of-distribution words.

They tested character n-grams from 2 to 6 characters and concluded that the optimal choice depends on the task and language. Longer n-grams are important to catch full morphemes and compound words, shorter to catch affixes. Shorter character n-grams could be a good resolution if you would like to study morphology more in detail.

There is also some new research out from Meta AI that suggests that entropy-based byte sequence patching could replace some tokenization techniques. Although I didn't look into it that deeply, the premise seems very interesting: when tokenizers are computed separately from the language model itself, all tokens are treated the same regardless of the fact that some tokens are harder to predict than others. A blog-post about the subject reminded me that indeed the first token is harder to predict than the last one. It makes so much sense to encode tokens on the basis of their information content, rather than some seemingly arbitrary token chopping schemes. It's also conceptually simpler than something like BPE, which also warms my heart. My intuition always leans towards treating tokenization more as encoding than compression, but of course it is likely problem-dependant which one to prefer.

Gemini and prompting

LLMs output probability distributions of a word given other words, and prompting is a way to condition the probability distribution such that the output corresponds to some particular domain. Prompt engineering is a field that aims to enhance our capabilities for extracting knowledge out of LLMs and there are a few common types of prompting that are being used.

- **Zero shot prompting**, where you just ask the model something, without examples
  "What is the meaning of life"
- **Few-shot prompting**, where you give an example of the desired output
  "Answer with one word. I like croissants: positive; I enjoy coffee:"
- **Chain of thought prompting**, where you provide the model with intermediate answers where you explain the type of reasoning you are looking for.

The last one was least familiar to me, so I looked into it a little bit. Sure, it can work, but in domains where you are not an expert and don't already know the answer, its practically impossible to evaluate the truthfulness of the output.

Chain of thought prompting. If your prompt contains intermediate answers where you explain the kind of reasoning you are looking for, it can help making the answers more correct, but this can only be evaluated in situations where you know the correct anwser, which kind of defeats the purpose of using it. I skimmed the Wei et al paper and it directed me to this other one from Talmor et al, which is a benchmark used in determining the usefulness of this type of prompting. All I'm saying is, if AMT workers are classifying something as abstract as *reasoning* with 90-95% accuracy, I think we can safely conclude that the reasoning that is being evaluated is not very complex reasoning at all.

Paraphrasing and content analysis

This was also one particular aspect of genLLMs I was interested to test. I started with going through the notebook as is, analysing the

strengths and weaknesses of AI-related research articles and paraphrasing them. At a first glance everything seems to be working quite well. My question was: where do these analyses of strengths and weaknesses come from? Because the way we talk about these technologies subtly implies that they could do this by “drawing from the vast knowledge embedded in the training data”. I fed it some articles that did not specify any strengths and weaknesses and behold: the LLM didn’t extract what was not there. This was an interesting finding.

I did not intimately know these papers, so I tested it with some articles I have written myself. It just re-stated what I had myself written as being the strengths and weaknesses of the paper. This confirmed my suspicion that the LLM does not actually reason in the way I expected, but it just extracts strengths and weaknesses from the paper if they are present, as they are written by the authors. The “problem” then i suppose is, there is a mismatch of expectation and reality, which I am sure is not just my subjective and personal experience. If you tell a system to “state the strengths and weaknesses” my mind automatically interprets this as “analyse the papers based on your knowledge of the subjects”. And by analyse, I mean in a critical, scientific sense. The anthropomorphing of these systems really hides the working principles behind these systems.

Why it works so well in this context, I would argue is because the convention in academic writing to specify some strengths and weaknesses of the research is so established, that it is in almost all papers. The problem of course is that it is very likely that the research contains errors and faults that the authors have not recognised. Also the strengths of any paper, as stated by its authors, are modified by a lot of things, not least catering to the target journals and possible future job prospects etc.

In any case, as a paraphrase machine, it seems to be reasonably good, with the caveat that in some writing communities it is a bit redundant, perhaps academic papers are not the best example to use here as they already contain a paraphrased version, the abstract. I suppose it is alright then, for getting a sense of what the author of the article said. Not so much that you could think of the LLM as analysing it in the broader context of “all science”.

It seems like the biggest problems come from what are our

understanding of the language used in the prompts.. stuff like “Summarize the text highlighting its strengths and weaknesses” Is something that is generally done by a human intelligence that uses its knowledge base to critically evaluate the strengths and weaknesses of an article and comparing it to other information. LLMs are incapable of this naturally, but the prompts we use affect the way we think about the technology. I think chatbots are the worst platform to use llms in because of this very reason.

Fine tuning

We also fine-tuned a language model on an instruction-followup dataset. To my understanding, instruction tuning is done to allow the model to learn the difference between a question and an answer, such that the model can be directed to make more answer-like continuations of the query sentence.

Fine tuning on ai generated text feels a bit odd, although it is being successfully done at a large scale currently. It might still work now and for some forecoming years, but as we now know, machine learning systems degrade quite a bit over time when trained on machine generated data. I wonder how we will solve this issue in the future. I ran into a paywall with Colab and left this part largely undone, although I did go through the code.

RAG

LLM outputs are designed to -look- right. In RAG, we restrict the “generation space” of the LLM to the documents we provide it. The RAG system, that has a search and a generation component, then can search the narrow dataset for passages that best correspond to your search query, or question and generate text based on it. Even the pure retrieval pipeline of this feels like it could be super useful. Again, I had barely time to go through the default notebook and as the end of year got busier, I first ran out of compute, then bought the pro version of Colab, but even the pro version does not give you sessions that would persist without constantly having to be active on the computer so I simply ran out of time to work on this. I’m used to leave stuff training overnight and I couldn’t figure out a way around this and didn’t want to actually start paying Google more money to be able to

complete the assignments. I hope future iterations of the course have some CSC compute allocated. I had this idea of building a RAG system for my Zotero library, such that I could ask my library stuff, the implementation will now happen outside the boundaries of the course but anyways, thank you for the course, I learned a whole lot and so did some of my friends by extension.