ECE 1786 Lecture #9

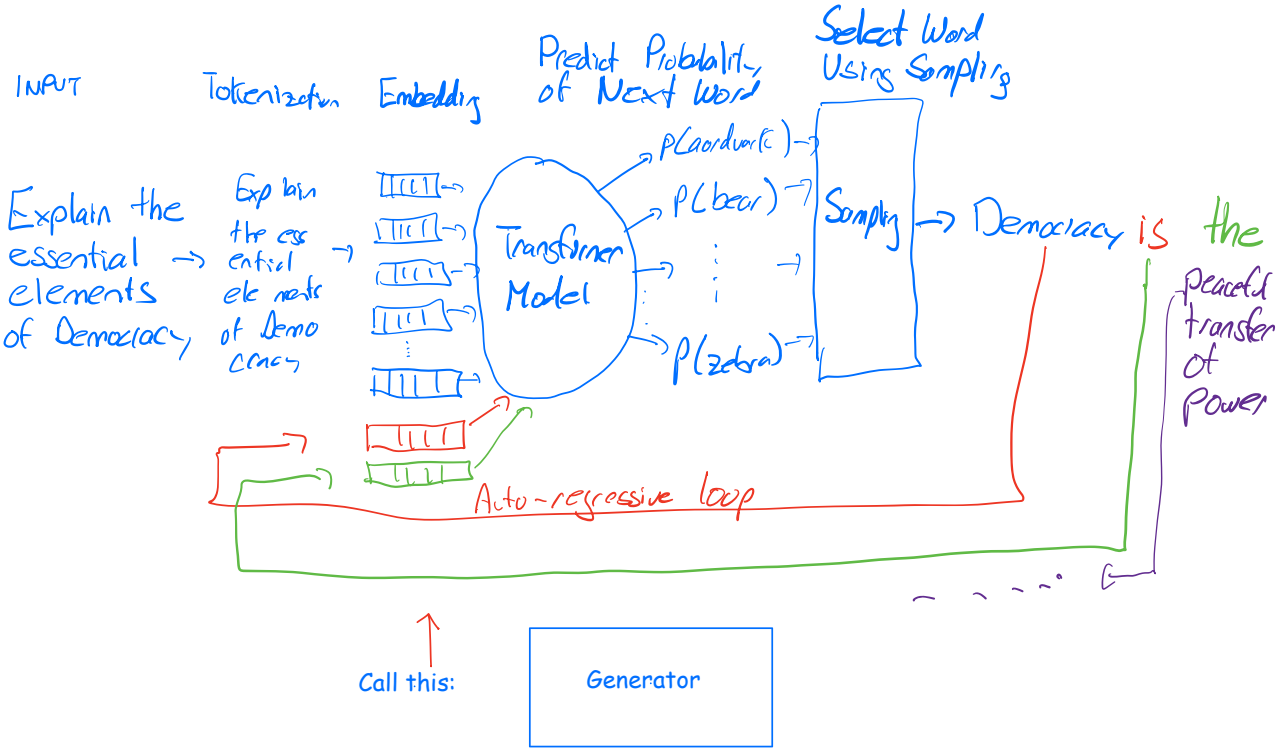
Work-in-Flight:

- Assignment 4 - Generation Probability Trees, Prompt Engineering and Agentic Systems; due November 13 (later than usual in the week)
- Project interim report - due Monday November 18 at 9pm
- Need to submit code to Github; I need your GitHub IDs, you must fill out this form: https://forms.office.com/r/0N683MR6ec

Last Day: Project Proposals! (Lecture 8++)
Before That: Scaling/Prompt Eng/COT/RAG (Lecture 7 on Video)

Today: How Large Language Models Became Good at doing what is asked: Reinforcement Learning from Human Feedback (RLHF) and Direct Preference Optimization (DPO)

- In lectures 4,5 and 6 we covered the architecture of a transformer, the process of training it, and then using it for text generation. Lecture 7 discussed how to prompt a transformer generator that has been well-trained. When you put it all together it has some key moving parts:



I did leave out one thing about that training that is important: What makes the LLMs good at doing what you ask them to do? Implied that training a model to predict the next word is all that is needed, but isn't true: there is a second layer of training. This is also used to put in 'guard rails' to prevent harm.

That said, I do believe (and there is general agreement) that the higher-level comprehension apparent in LLMs does come from this first level training, called pre-training against trillions of tokens. (llama3 405B using 15.6T tokens!)

However, if you were to use a model that is only pre-trained in this way, you would find that its output would be messy and unsatisfactory in many ways - poor output (random characters, weird repetition and off-topic answers) would occur often, and the model's ability to comprehend what you wanted would be worse than you've experienced with chatGPT and GPT-4o.

- I'll post an example query that shows this of what every other model looked like this previous to GPT-3.5, the original Nov 2022 chatGPT

OpenAI led the way on this second layer of training.They call it "Reinforcement Learning with Human Feedback." (RLHF). An alternative has been proposed and is gaining traction, called Direct Preference Optimization (DPO). Start with RLHF.

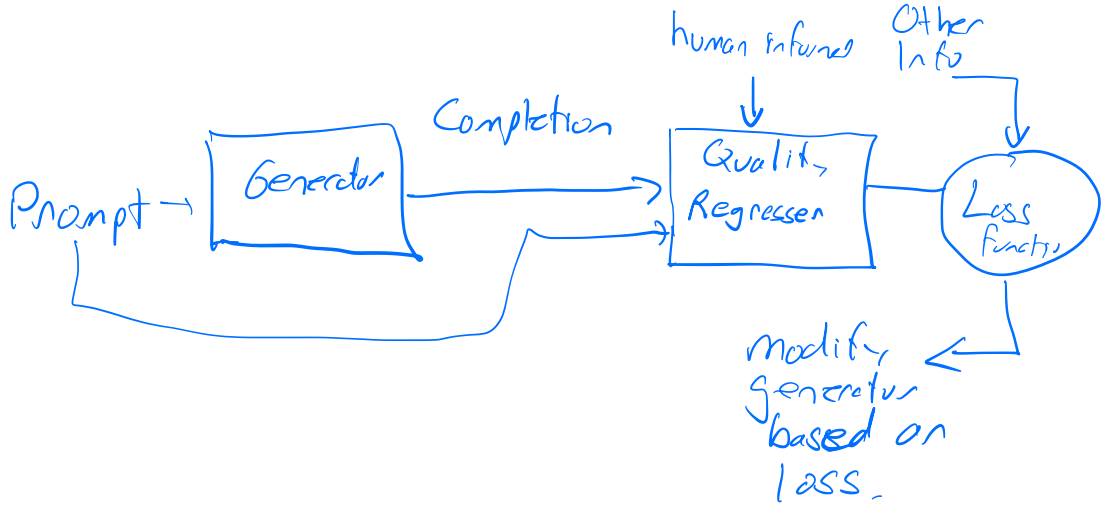
- It is possible to describe the essentials of what is going on first without using the structure of Reinforcement Learning (RL)
-
- The core of the second layer RLHF is described in a paper by Ouyang et. Al, (https://arxiv.org/abs/2203.02155 also posted in the course notes) but also described in a more readable, higher-level form in a Huggingface blog post (https://huggingface.co/blog/rlhf that is also posted).
- The Llama2 paper from Meta also speaks to some interesting issues around RLHF and safety vs. quality the Llama2 training paper (https://arxiv.org/abs/2307.09288, also posted).
- Note the LLama3 paper - https://arxiv.org/abs/2407.21783
  - Meta/Llama is the only one telling us inner details these days

OK, so Let's describe the extra layer of training!

Before we begin, let's define a little terminology: A prompt is the text that is input to the LLM. A completion is the full sequence of text produced in the auto-regressive loop - i.e. the sequence of tokens that are formed into words.

Here is a summary of the method used to make the models better, after the extensive pre-training on very large corpora to predict the next word:

1. Pay humans to create a dataset of prompts + completions (questions/answer; or requests/responses) by humans, which do a good job of answering/ responding to well formed questions/requests. Pay people to do this, and train them well (Meta/llama2 paper says this quality really matters)
2. Fine tune the model, (predicting the next word) on these specific examples
   A. improves the model some, but isn't the core method
3. Build and train a classifier (really a regresser) that produces a rating of the quality of these (prompt+completion)s - rate completion given prompt
4. Use the output of the classifier as part of a loss function that judges the completion given the prompt. It is used as a reward function in the reinforcement learning-based optimization of the generator. That is, this loss (plus another) is used to adjust the parameters in the Transformer so that the probabilities generated ultimately make better 'completions' that humans prefer.



The details are interesting, with some complexity. One of the most clever parts is that once you've got the classifier and this system going, it is possible to automatically generate many more data examples (of prompts/completions) to train on.

Step 1: Hire Humans to create good examples of prompt+completions. E.g, from Ouyang paper, both prompt & "labeler demonstration" were human-produced:

Prompt:
Create a shopping list from this recipe:
Trim the ends off zucchini. Cut zucchini in half lengthwise; scoop out pulp, leaving 1/2-in. shells. Finely chop pulp. In a skillet, cook beef, zucchini pulp, onion, mushrooms and peppers over medium heat until meat is no longer pink; drain. Remove from the heat. Add 1/2 cup cheese, ketchup, salt and pepper; mix well. Spoon into the zucchini shells. Place in a greased 13x9-in. baking dish. Sprinkle with remaining cheese.

Labeler demonstration
ucchini, beef, onion, mushroom, peppers, cheese, ketchup, salt, pepper

Jumping ahead & reiterating why this step is important, we can see what the non-RLHF'd "GPT3 175B" does with the prompt, and what the the RLHF'd one (called "instruct GPT" back then). The first is useless, the second answers the question.

GPT-3 175B completion:
Bake, uncovered, at 350° for 20-25 minutes or until zucchini is tender and cheese is melted.

InstructGPT 175B completion:
Bake, uncovered, at 350° for 20-25 minutes.

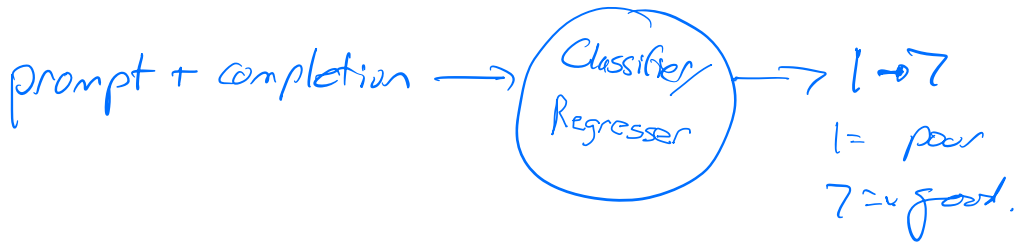
- 8 zucchini squash, ends trimmed
- 1 pound ground beef
- 1 cup finely chopped zucchini pulp
- 1 small onion, chopped
- 8 ounces diced mushrooms
- 1 sweet red or green pepper, diced
- 1/2 cup grated cheese
- 1/4 cup ketchup
- Salt and pepper to taste
- 1 cup shredded cheese

Figure 49: Labeler-written prompt from our dataset, along with the human-written demonstration, and completions from GPT-3 175B and InstructGPT175B. Prompt is lightly cherry-picked (5 selected from 15 to show a diverse range of tasks), and the completions are not cherry-picked.

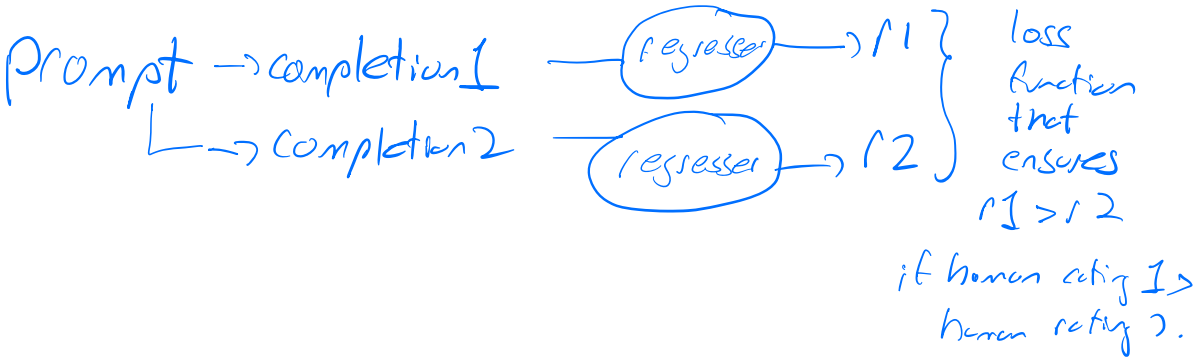
See the Ouyang paper for many examples of this, in the appendices.
A total of 12,000 prompt+completions were done by humans for pay
At first, these were used to do regular 'fine-tuning' - they were used as training input to predict-the-next-word, as you're familiar with from Lecture 5/6.

- Call this "Supervised Fine Tuning" or SFT (still thought to be useful)

Step 2: Train a classifier/regresser model that takes in a prompt+completion and produces a rating of the quality of the completion given the prompt. The rating is on a scale of 1-7. Recall: What is a regresser?



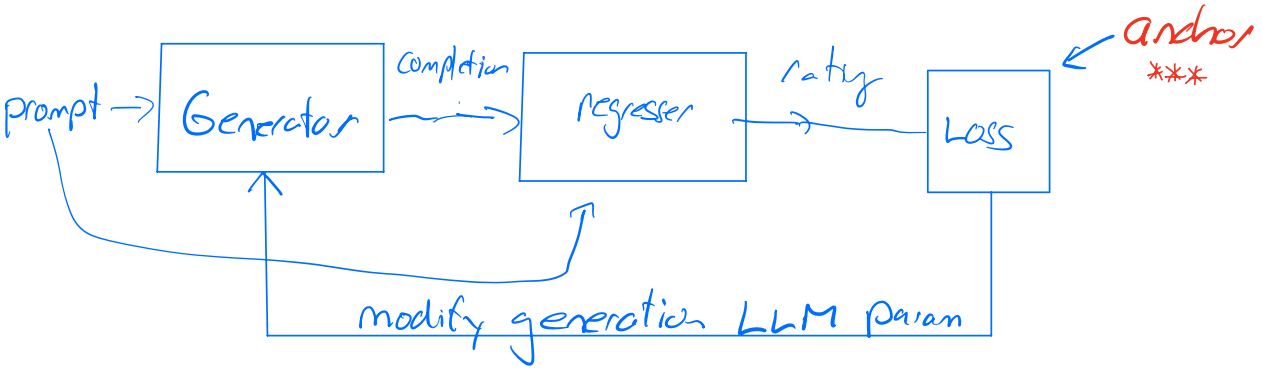
- To do this, collect human labels on a set of prompts + completions - again paying people
  - issue: humans are calibrated differently, one person's 1, is another's 5 (*)
  - So, the regresser/classifier is trained in a clever way to account for this: present 2 completions for the same prompt;
  - Ask same human to label both completions with on scale 1-7
  - The goal of the regresser training is to make the model's rating such that it prefers the ones that humans prefer. It is not to make the ratings match, because of (*).



- Notice that we can get as many completions as we want, by simply re-running the generator (diff seed). That's how we get completion1 and completion2 (**)
- Aside: what sort of model should we use to make the regresser?
  - Answer - a Transformer, perhaps even the same model as the LLM being, with the language head is replaced with a single regression output
  - OpenAI used small GPT-3; Meta in LLama2 used its largest Llama2

9-6

Step 3: Create a larger dataset of prompts + completion1, 2, 3 .... N (i.e. many completions for a given prompt, using **) Use those as training data for a new round of training of the generator, that makes use of the step 2 classifier:



- Can generate many completions, as said, so lots of potential data!
- How is the quality of the regresser used in gradient descent?
  - each token that was generated for a completion that is considered good
    - is given 'good' loss (-log(that token)) - that token is encouraged
  - each token for bad completion is discouraged (+log(that token))

- But that's not all: This approach, by itself will cause the generator to lose its basic knowledge, and the generator will become stupid.
  - The regresser human preference goal will override the basic knowledge

- Solution: the generator output logits/probabilities need to be 'anchored' to that to the original training/knowledge, while they are also being 'pulled' by this new loss to create a good quality human preferred output

Loss = f(Regresser(prompt+completion), model outputs of Unchanged SFT)

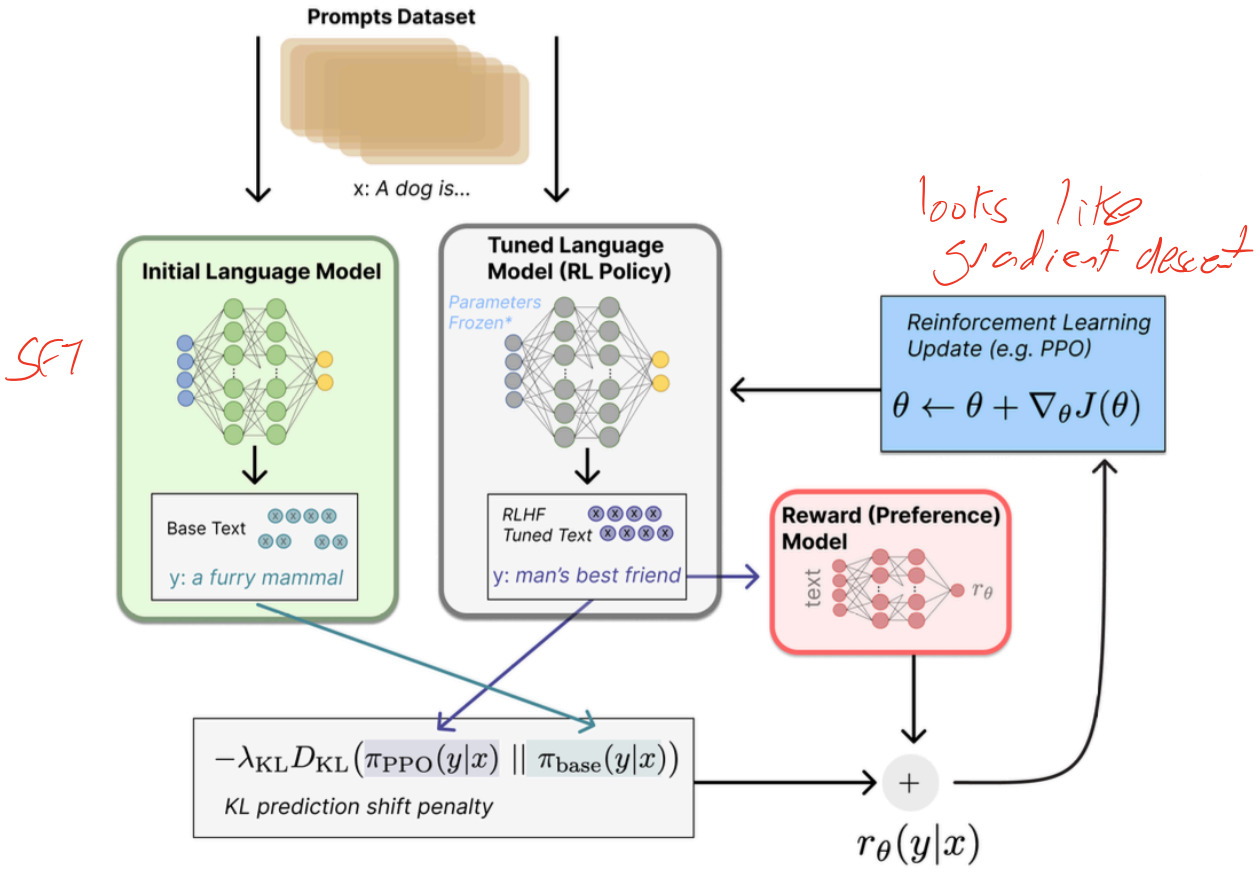
"Supervised Fine Tuning"

- The unchanged SFT is the model after step 1.
- Take all the logits of SFT, and the logits and of the generator, and force them to stay close. (called the K-L divergence)

The picture below, from the Huggingface blog, gives some sense of the whole process. It makes use of terminology from Reinforcement Learning that I have not used.

  - "Policy" = the generator language model being tuned;

9-7



- "Reward Model" = the Regresser that was trained to figure out how good a prompt+completion is

ECE 1786 Project Progress Report (Bring up Quercus Assignment)

- Due on Monday November 18
- Hard word limit - 1000, with penalty
- Goal is to make sure you're pushing for progress!
- See the assignment on line for the deeper description