CSC-496/696: Natural Language Processing and Text as Data

Lecture 21: Finetuning Generative LLMs and Applications

Patrick Wu

Tuesday, November 19, 2024

Lecture Contents

Announcements

Assignment 5

- Assignment 5 due on Tuesday, November 26 (one week from today)
- My suggestion: get it done as soon as possible to give yourself time to complete your final projects
- It might also be helpful for completing the final project (if you're using a generative LLM)

Final Project

- Reminder that you can still sign up for a 15 minute meeting about the final project
- Presentation schedule was posted on Canvas
- 5 minute presentation for individual projects
- 10 minute presentation for group project (expectation is that each person speaks 5 minutes)
- 5 minute Q&A
- Small amount of extra credit if participate in Q&A both days

Questions?

Review of Finetuning

Finetuning LLMs

- Finetuning: process of taking a pretrained language and further training it on a domain-specific dataset

Finetuning LLMs

- Finetuning: process of taking a pretrained language and further training it on a domain-specific dataset
- All finetuning is a form of **transfer learning**, which is when we take a model trained on a one domain and apply it to another domain that it was not trained on

Finetuning LLMs

- Finetuning: process of taking a pretrained language and further training it on a domain-specific dataset
- All finetuning is a form of **transfer learning**, which is when we take a model trained on a one domain and apply it to another domain that it was not trained on
- Finetuning comes after pretrianing

Finetuning LLMs

- Finetuning works because what the model learned from being pretrained on such a large amount of text is not exclusive to that body of text

Finetuning LLMs

- Finetuning works because what the model learned from being pretrained on such a large amount of text is not exclusive to that body of text
- In other words, what the model learns about grammar, syntax, semantic meaning, etc. also carries over to completing a task related to social media posts, even if those posts are brand new and the LLM has not been pretrained on it

Finetuning LLMs

- Finetuning works because what the model learned from being pretrained on such a large amount of text is not exclusive to that body of text
- In other words, what the model learns about grammar, syntax, semantic meaning, etc. also carries over to completing a task related to social media posts, even if those posts are brand new and the LLM has not been pretrained on it
- Unlike prompt engineering approaches and in-context learning, finetuning actually makes updates to the model's parameters using gradient-based approaches

Finetuning LLMs

- Finetuning works because what the model learned from being pretrained on such a large amount of text is not exclusive to that body of text
- In other words, what the model learns about grammar, syntax, semantic meaning, etc. also carries over to completing a task related to social media posts, even if those posts are brand new and the LLM has not been pretrained on it
- Unlike prompt engineering approaches and in-context learning, finetuning actually makes updates to the model's parameters using gradient-based approaches
- The amount of data required is much less to achieve comparable results compared to training a model from scratch

Finetuning LLMs: Adding Classification Heads

- We could, in theory, add a *head* to the model

Finetuning LLMs: Adding Classification Heads

- We could, in theory, add a *head* to the model
- This head is used to process the final output hidden state (or the collection of hidden states) for our specific task

Finetuning LLMs: Adding Classification Heads

- We could, in theory, add a *head* to the model
- This head is used to process the final output hidden state (or the collection of hidden states) for our specific task
- We can take the outputs of the LLM, pool them in some way (such as averaging them), and then project this pooled embedding down to the number of classes that we're interested in

Finetuning LLMs: Adding Classification Heads

- We could, in theory, add a *head* to the model
- This head is used to process the final output hidden state (or the collection of hidden states) for our specific task
- We can take the outputs of the LLM, pool them in some way (such as averaging them), and then project this pooled embedding down to the number of classes that we're interested in
- Then, use backpropagation to finetune the entire model

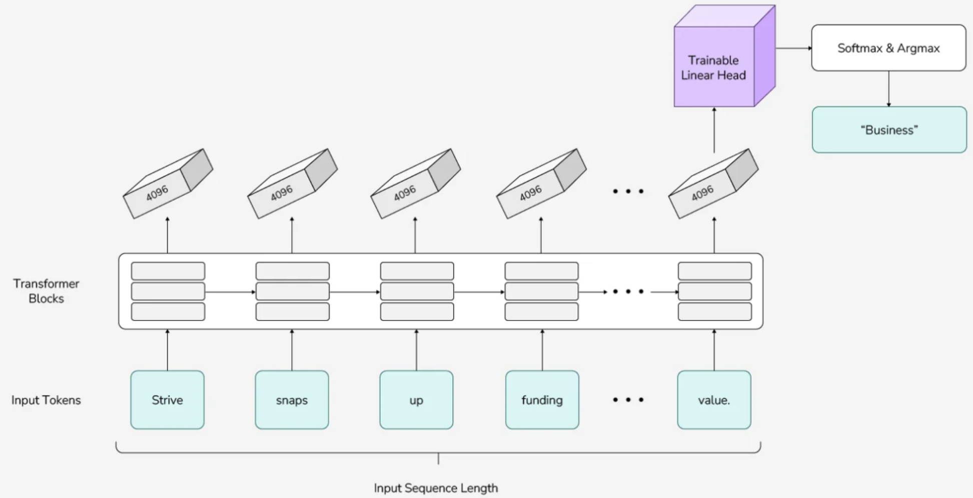


Figure from Strive Works.

Common Pitfalls of Finetuning for Classification

- There is often the threat of overfitting when finetuning your model
  - We often use finetuning when we don't have a lot of data
  - But it is easy to over-finetune the model on this small amount of data, reducing generalizability of the models

Finetuning LLMs: Supervised Finetuning (SFT)

- Supervised finetuning (SFT) is often used to finetune generative LLMs
- Although generative LLMs can be finetuned using a specialized head in the same way we would finetune a model like BERT, SFT is often more effective
- SFT: another approach to finetuning a pretrained LLM using labeled data
- Usually, we use next word prediction
- We can either use next word prediction on the entire input-output sequence, or we can use next word prediction only on the outputs using the inputs as context

Finetuning LLMs: Supervised Finetuning

- We saw that SFT is used with instruction finetuning
- Input: instruction
- Output: a response to the instructions (such as a completed task or an answer to a question)
- Used to finetune a generative LLM so it follows instructions well

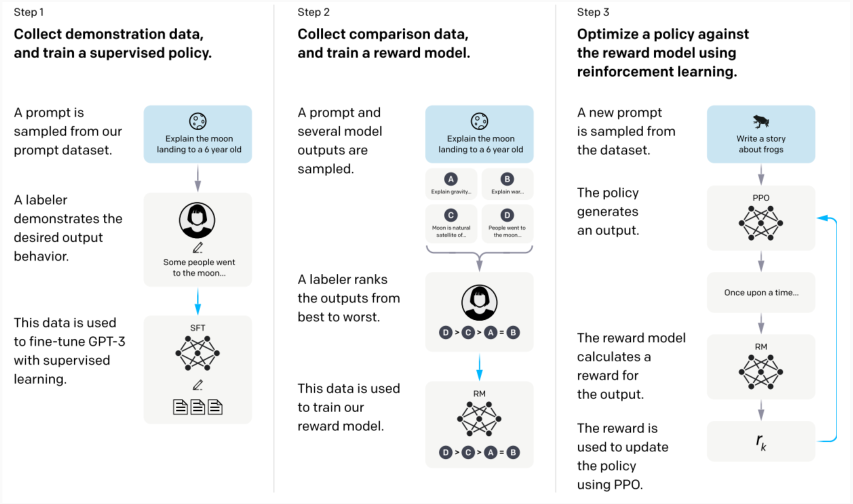

Step 1
Collect demonstration data, and train a supervised policy.
A prompt is sampled from our prompt dataset.
Explain the moon landing to a 6 year old
A labeler demonstrates the desired output behavior.
Some people went to the moon...
This data is used to fine-tune GPT-3 with supervised learning.
SFT
Step 2
Collect comparison data, and train a reward model.
A prompt and several model outputs are sampled.
Explain the moon landing to a 6 year old
A Explain gravity...
B Explain war...
C Moon is natural satellite of...
D People went to the moon...
A labeler ranks the outputs from best to worst.
D > C > A = B
This data is used to train our reward model.
RM
D > C > A = B
Step 3
Optimize a policy against the reward model using reinforcement learning.
A new prompt is sampled from the dataset.
Write a story about frogs
The policy generates an output.
PPO
Once upon a time...
The reward model calculates a reward for the output.
RM
The reward is used to update the policy using PPO.
r_k

Diagram from Ouyang et al. (2022).

Review Question

Finetuning requires updating all the weights of the model.

(A) True

(B) False

Lab from Last Class

Here is a link to the lab that will show how we can finetune RoBERTa:
https://colab.research.google.com/drive/1ldNbrEtuwqiEvTCGoXv-hafOFwL_DrLm?usp=sharing

But wait...ChatGPT is so powerful! Why use encoder-only models?

- Encoder-only models tend to be **much** smaller than models like GPT
- Generative (decoder-only) LLMs also tend to be much slower because of the generation process
- Encoder-only models that are finetuned on a large number of hand-labeled data can be *much* faster than GPT
- If we are applying an automated classification model to a large number of observations (e.g., one of my projects has several hundred millions of tweets that need to be classified), encoder-only models are much more efficient than generative LLMs

Combining Decoder-Only and Encoder-Only Models

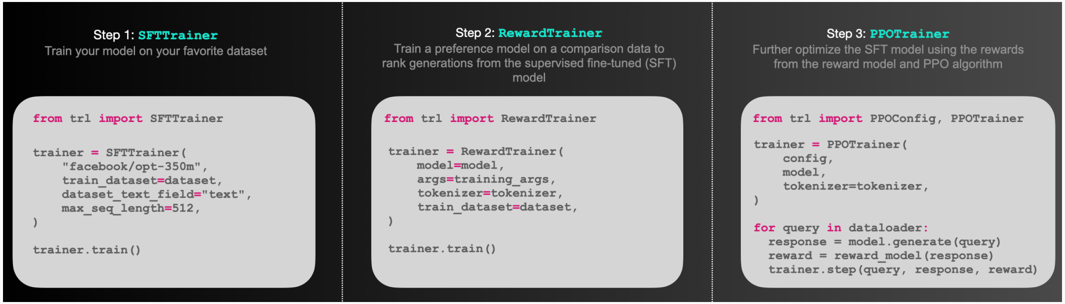
- Ongoing research on using generative LLMs to generate high quality labels, and then using this to finetune an encoder-only model
- Research on using humans to label only hard-to-label observations, and using generative LLMs to label “easy” or “obvious” observations

Finetuning Generative LLMs using SFT

Hugging Face’s `trl` Library

- Hugging Face has recognized that many people might want to use SFT and RLHF with their own data
- They now have a library called `trl`, or **T**ransformer **R**einforcement **L**earning
- You can read more about it here: https://huggingface.co/docs/trl/en/index
- You can use this to build your own GPT-like model

trl



Step 1: SFTTrainer
Train your model on your favorite dataset

```
from trl import SFTTrainer

trainer = SFTTrainer(
    "facebook/opt-350m",
    train_dataset=dataset,
    dataset_text_field="text",
    max_seq_length=512,
)

trainer.train()
```

Step 2: RewardTrainer
Train a preference model on a comparison data to rank generations from the supervised fine-tuned (SFT) model

```
from trl import RewardTrainer

trainer = RewardTrainer(
    model=model,
    args=training_args,
    tokenizer=tokenizer,
    train_dataset=dataset,
)

trainer.train()
```

Step 3: PPOTrainer
Further optimize the SFT model using the rewards from the reward model and PPO algorithm

```
from trl import PPOConfig, PPOTrainer

trainer = PPOTrainer(
    config,
    model,
    tokenizer=tokenizer,
)

for query in dataloader:
  response = model.generate(query)
  reward = reward_model(response)
  trainer.step(query, response, reward)
```

From Hugging Face's website

`SFTTrainer`

- In the lab that we'll turn to in a second, we'll look at `SFTTrainer`
- By default, it will continue the next token prediction task
- It will handle many of the "complicated" preprocessing tasks under the hood, although it is always good to know what kind of preprocessing is happening
- For example, it will handle tokenization and batching for you

Low-Rank Adaptation (LoRA)

- Method to finetune LLMs, especially *very* large LMs, efficiently
- Developed to reduce the computational and storage costs of finetuning these models

LoRA

- Finetuning LLMs can be computationally expensive and requires significant GPU memory
- Modern generative LLMs, in particular, are quite GPU memory-hungry
    - Difficult to run models such as Llama-3.1-70B, which has (as the name implies) 70 billion parameters
    - Even harder to finetune these models
- LoRA focuses on adapting only a small number of parameters using *low*-rank matrices, reducing resource requirements

What is Rank?

- The *rank* of a matrix is the maximum number of linearly independent rows or columns
- It indicates the “complexity” or amount of information the matrix can represent
- In LoRA, using low-rank matrices means representing information in a simpler, more efficient way

LoRA

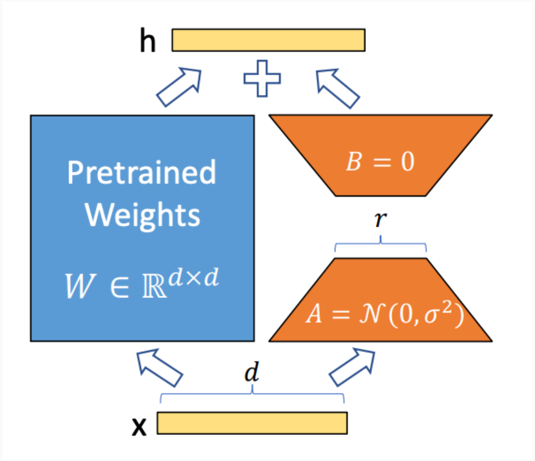

h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

Diagram from LoRA paper

LoRA

- Let W_0 be the pretrained weight matrix
- $W_0 \in \mathbb{R}^{d \times k}$
- We can constrain the update by representing the weight matrix using a low-rank decomposition: $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$
- Here, $r \ll \min(d, k)$
- Then,

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

Hyperparameters for Using LoRA

- r: the rank of the matrix. Notice that this should be quite low (popular numbers include 8, 16, etc.)
- α: a scaling factor introduced to control the contribution of the low-rank update BA to the original weight matrix
- Used to balance the contribution of BA
- The pretrained weights W are often much larger in scale compared to the low-rank adaption BA
- There's no hard and fast rule about how to choose r and α, but usually, $\alpha > r$
- In the lab, I set $\alpha = 32$ and $r = 16$

Lab

Link to the finetuning generative LLMs using SFT lab:
https://colab.research.google.com/drive/1ipqNAXfxqnKiktKBdSOVKMg0_LyHqasK?usp=sharing

OpenAI API

OpenAI API

- While it is easy to use `chatgpt.com`, it is often inefficient to type in a prompt and then manually copy-paste the generated response one-by-one
- However, OpenAI's model is not open, meaning we cannot load it via Hugging Face
- We interact with it using an *API*

What is an API?

- API: Application Programming Interface
- We can send requests to a *server*; the server can then send a response to us (the *client*)
- Here is an overview of the OpenAI API: https://platform.openai.com/docs/overview

OpenAI API

- Provides an interface to access their models
- You will need to sign up for an OpenAI account and create an API key
- This key will be used to connect your OpenAI account with the API
- You will also need funding in your account
- Pricing can be found here: `https://openai.com/api/pricing/`
- Note that each model costs different amounts, and are usually charged on a per-million tokens basis

OpenAI Models

- GPT-4o: most advanced model *and* multimodal; has 128k context
- GPT-4o mini: most cost efficient model and has vision capabilities; has 128k context
- OpenAI o1-preview: a new reasoning model for complex tasks; *really* expensive
- OpenAI o1-mini: a faster reasoning model but still pretty expensive (more expensive than GPT-4o)

Finetuning Models

- You can also use OpenAI's API to finetune their models
- You can't finetune using specialized heads, but you can finetune using SFT
- Can be quite costly even to use the finetuned model, but often useful if a task you're interested is better learned through "showing" rather than "telling"

OpenAI API Lab

Lab here: https://colab.research.google.com/drive/1eIgKq-0rWmqFHekwWuZwJ7EVu1SKtUsj?usp=sharing

Discussion: What are some of the advantages and disadvantages of blackboxing away a model like this?