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

Lecture 20: Prompt Engineering and Finetuning

Patrick Wu

Friday, November 15, 2024

Lecture Contents

1. Announcements
2. Review of Prompt Engineering and Parameters
3. RAG Lab
4. Finetuning

Announcements

Assignments

- Assignment 3 graded and returned
- Assignment 5 released: consists of one question
- Originally due on Friday, November 22, but I'll extend the deadline to Tuesday, November 26
  - Feel free to use late days
- But just a reminder that your final project presentations are a week after Thanksgiving, so I would encourage you to get it done sooner rather than later

Final Project

- I sent out an email with some feedback on the project proposal
- I also included a link to sign up for a time for a 15-minute meeting
- Of course, I can't force you to meet, but I highly recommend it!
- A chance before Thanksgiving to review the data, ask questions about methods, and ask questions about what the presentation and report should look like

Final Project Presentations

- 5 minutes for solo projects, 10 minutes for group projects
- I suggest you use slides or some kind of visual
- Will announce on Canvas which day you will be presenting on
    - Will be randomly determined

Questions?

Review of Prompt Engineering and Parameters

Prompting

- When using generative LLMs, there are many different ways to develop prompts to use
- Zero-shot prompting, few-shot prompting, chain-of-thought, etc. are all designed to extract knowledge from the LLM without any kind of finetuning
- In other words, the model's parameters are fixed

Prompting

- Given the prompt as context, the LLM will generate the next token based on its token probability, conditioned on the prompt: $P(w_i|w_{<i})$
- **In-context learning** is when the model learns information from the prompt even though there are no gradient-based updates to the model's underlying parameters

In-Context Learning

Define happy as "feeling or showing sorrow." Is this statement happy?

"Wow, I wish I wasn't here right now."

Given the redefined meaning of *happy* as "feeling or showing sorrow," the statement "Wow, I wish I wasn't here right now" does indeed seem *happy*. The tone reflects discontent or sadness, which aligns with the redefined sense of *happiness* as sorrow.

Two Parameters to Control

- **Top-p Sampling**: sample words that fall among the top p cumulative probability
- **Temperature**: if $0 \leq \tau < 1$, makes probability distribution spikier; if $\tau > 1$, makes probability distribution flatter

Two Parameters to Control

- In both cases, we're trying to ensure that we get high quality words *and* a diversity of words
- Don't want model to always just choose the token with the highest probability
- We can do this if we set the temperature to 0
  - Called *greedy decoding*
  - Can often lead to repeated text generation

Dynamic Prompts

- Ideally, we would have dynamic prompts: a prompt that evolves and is suited for each specific instruction or task in each prompt
- For example, if a question is asked about a topic, we would ideally like the model to access the Wikipedia page for that topic
- This can reduce *hallucinations* from the model

Retrieval-Augmented Generation (RAG)

- A framework that allows us to include examples/demonstrations/external information dynamically
- We will have a database containing the information that we would potentially want to include in the prompt, and the prompt itself (often called a query)

Retrieval-Augmented Generation

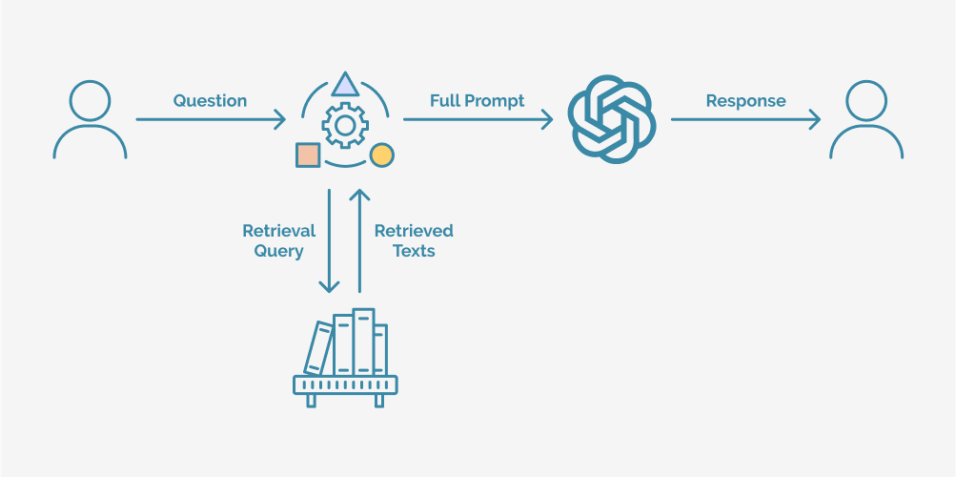

Diagram from Trantor

SentenceBERT (SBERT)

- SBERT is often used in RAG
- It calculates sentence embeddings (AKA document embeddings) of natural language input
- For example, it will map the sentence "I like turtles." to an embedding
- Dimension of the embedding depends on which model you use

SBERT

- These BERT-based embedding models are often used out of the box, meaning that we don't finetune these models at all
- There are also ways to finetune these models for task-specific similarity tests
- But often used without finetuning

RAG

1. Map the original prompt (the query) to an embedding using SBERT
2. Map all database documents to embeddings using SBERT
3. Use some kind of similarity measure, such as cosine similarity or L2 distance, to find the database documents closest to the query
4. Input all database documents closest to the query in a full prompt along with the original query
5. Input this into a generative LLM of your choice

Advantages of RAG

- More computationally efficient: because we are not finetuning the model, this is often more computationally efficient and thus more cost-effective
- Allows you to update LLMs quickly: it might take a long time to update an LLM with the latest information, which is why companies such as OpenAI and Anthropic do not often update the cutoff date of their models
- Potentially more interpretable: because the model is pulling information from documents that *we* control, this can potentially make the solutions generated by an LLM more interpretable

Disadvantages of RAG

- Dependency on the Retriever Quality: in our case, we're dependent on SBERT actually being a good way to calculate sentence embeddings as a way to get us good documents
- Does not make up for a model that struggles in pattern recognition: if a model struggles to recognize patterns in whatever application we're interested in, this can't be solved using RAG

RAG Lab

We'll now turn the lab, which can be found here.

Finetuning

A Note Before We Dive Into Finetuning

- It's okay if you don't really understand what is going on after this point
- There is no easy way to discuss these topics, but I will go light on the technical details
- The learning objective, instead, is to become familiar with some of these terms and overall concepts
- If you decide to pursue NLP further, these terms may appear quite often

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

Encoder-Only LLM: A Quick Recap

- Recall that an encoder-only LLM will always output the same number of vectors as there are input vectors
- So if we input $x^{(1)}$, $x^{(2)}$, and $x^{(3)}$, we will get three output vectors $z^{(1)}$, $z^{(2)}$, and $z^{(3)}$
- We can, for example, average the vectors and then input the averaged vector into a single layer fully connected neural network and project it down to, say, 10 dimensions (for 10 classes, if this is a 10-class classification problem)

Decoder-Only LLM: A Quick Recap

- Similarly, a decoder-only LLM will also always output the same number of vectors as there are input vectors
- So if we input $x^{(1)}$, $x^{(2)}$, and $x^{(3)}$, we will get three output vectors $z^{(1)}$, $z^{(2)}$, and $z^{(3)}$
- But instead of using $z^{(1)}$ to predict the next word, we can, again, average the vectors and input the averaged vector into a single layer fully connected neural network and project it down to, say, 10 dimensions (for 10 classes, if this is a 10-class classification problem)

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

Finetuning LLMs: Adding Classification Heads

- The head is usually randomly initialized and can be as simple as a fully connected neural network

Finetuning LLMs: Adding Classification Heads

- The head is usually randomly initialized and can be as simple as a fully connected neural network
- The weights of the head are then adjusted using backpropagation (along with the rest of the model)

Finetuning LLMs: Adding Classification Heads

- The head is usually randomly initialized and can be as simple as a fully connected neural network
- The weights of the head are then adjusted using backpropagation (along with the rest of the model)
- This *requires* a labeled dataset: you must know the labels for each observation in order to adjust the weights of this model

Finetuning LLMs: Adding Classification Heads

- The head is usually randomly initialized and can be as simple as a fully connected neural network
- The weights of the head are then adjusted using backpropagation (along with the rest of the model)
- This *requires* a labeled dataset: you must know the labels for each observation in order to adjust the weights of this model
- The head of the model differs from the body of the model, which is usually where broad, transferred language resides

Finetuning LLMs: Adding Classification Heads

- Using classification heads is a typical way to use encoder-only models such as BERT, RoBERTa, etc.
- Encoder-only models are particularly good for finetuning for classification tasks because they use bidirectional attention, meaning the model can consider both left and right context simultaneously
- The masked language modeling tasks ensures these models are optimized to understand context (MLM uses context to predict a masked out word)
- While it is possible to finetune a decoder-only model using this same approach, they often don't work as well

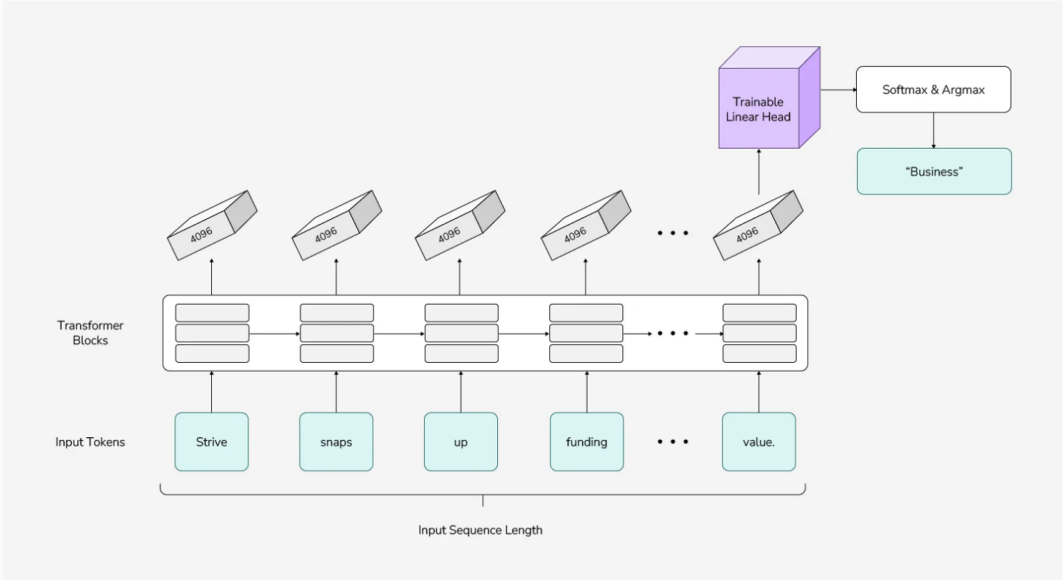

Figure from Strive Works.

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

Common Pitfalls of Finetuning for Classification

- Risk of *catastrophic forgetting*: it is common to see that a model that has been finetuned has actually lost some of its general language understanding that it acquired during pretraining
  - This can lead to poor generalizability of the model to out of sample data
  - A common remedy to catastrophic forgetting is to *freeze the underlying model*

Freezing the Underlying Model

- A common way to make finetuning more efficient is to freeze the underlying LLM
- For example, you might freeze all the parameters of BERT
- You only adjust the weights of the classification head, which may be a few-layer neural network
- Often much more efficient than finetuning the entire model, but may reduce performance

Finetuning Not Limited to Classification

- This kind of finetuning is not limited to classification.

Finetuning Not Limited to Classification

- This kind of finetuning is not limited to classification.
- You can also finetune a model to do, for example, entailment tasks well.

Finetuning Not Limited to Classification

- This kind of finetuning is not limited to classification.
- You can also finetune a model to do, for example, entailment tasks well.
- For example, you can give a model a premise and a hypothesis about that statement. Then, the model can classify whether the premise entails the hypothesis, the premise contradicts the hypothesis, or the premise does not entail nor contract the statement.

Finetuning Not Limited to Classification

- This kind of finetuning is not limited to classification.
- You can also finetune a model to do, for example, entailment tasks well.
- For example, you can give a model a premise and a hypothesis about that statement. Then, the model can classify whether the premise entails the hypothesis, the premise contradicts the hypothesis, or the premise does not entail nor contract the statement.
- Too many finetuning tasks for us to go through, but they all follow a similar formula: we use a head in order to complete a specialized task, and this head is separate from the body

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
- But instead of using next word prediction with the entire sequence, next word prediction only occurs in 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

Using RAG vs. Finetuning

- A common question: when do we want to use RAG vs. finetuning?
- It is often said that if you want the model to produce *better* information, it is often worthwhile to use RAG
- RAG allows us to control where the information comes from
- On the other hand, you might want to use SFT if you want the model to do better in responding to something

Example: Customer Support

I can't log into my account. What should I do?

If you're having trouble logging into your account, here are some steps to help you troubleshoot:

1. **Check Your Credentials**:
   - Ensure you're entering the correct username/email and password.
   - Be mindful of any extra spaces, capitalization, or special characters that might cause issues.
2. **Reset Your Password**:
   - Look for a "Forgot Password" link on the login page. Follow the instructions to reset your password. You may need access to the email associated with your account.

Review Question

Finetuning does not involve gradient updates to the parameters of the model.

(A) True

(B) False

Review Question

Finetuning using a specialized head can be used with both encoder-only and decoder-only models.

(A) True

(B) False

Lab

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