Lab-111 : LLM Prompting

1. Name : Ajeet Kumar
2. Linkedin : https://www.linkedin.com/ajeetkumar09
3. Berkely MOOC Course(2024) : Large Language Model Agents
4. Berkely MOOC Course(2025) : Advanced LLM Agents(currentely enrolled)

1. **What is Large Language Models(LLMs) ?**

> LLMs like Falcon, LLaMA etc are pretrained transformers models initially trained to predict the next token given some input text. LLM model size having several hundred billions of parameters and have been trained on trillions of tokens for an extended period of time on internet data and fraction of universe knowledge dataset. We can use these models to solve multiple NLP tasks out of the box by instructing the models with natural language prompts.

> Our aim to design prompt to ensure the optimal output. Prompt Engineering - "An iterative process that requires a fair amount of experimentation". We know that natural languages are more flexible and expressive than the programming languages therefore they can introduce some ambiguity. The prompts in natural languages are quite sensitive to changes because minore change can leads to quite different outputs. There is no best algorithm for creating prompt but researcher have figure out best practices for creating optimal results that are more consistent with better LLM prompts.

Prompting

Modern LLMs : Decoder-only transformers - Llama2, Falcon, GPT2

Encoder-decoder transformers - Flan-T5, BART

Encoder-decoder-style models are used in generative task where output approx totally dependent on input like translation and summarization etc.

Decoder-only transformer model are usefull for all others types of generative tasks.

We will use pipeline to generate text with LLM and each LLM might have different pipeline.

Most LLM checkpoints available in two versions : base and instruct on huggingface-hub like `tiiuae/falcon-7b` and `tiiuae/falcon-7b-instruct`.

What are Base Models ? Base models are pretrained llms which are excellent at completing the text when given an initial prompt, however, they are not ideal for NLP tasks where they need to follow instructions, or for conversational use.

What are Instruct Models ? Instruct Models are also pretrained model and when we finetune the base model on instructions and conversational data these result in checkpoints which makes them better choice for many NLP tasks according to instructions and conversational data.

```
!pip install -q transformers accelerate
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 3.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 46.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 41.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 33.7 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 19.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 6.3 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 86.5 MB/s eta 0:00:00
```

```
# depenencies
from transformers import pipeline, AutoTokenizer
import torch
```

```
# Text generation task with decoder-only models i.e running inferences from gpt-2 model
torch.manual_seed(0)
gen = pipeline('text-generation',
               model = 'openai-community/gpt2')
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public model
  warnings.warn(
config.json: 100%                                   665/665 [00:00<00:00, 40.4kB/s]
model.safetensors: 100%                             548M/548M [00:03<00:00, 155MB/s]
generation_config.json: 100%                        124/124 [00:00<00:00, 4.58kB/s]
tokenizer_config.json: 100%                         26.0/26.0 [00:00<00:00, 1.54kB/s]
vocab.json: 100%                                    1.04M/1.04M [00:00<00:00, 4.76MB/s]
merges.txt: 100%                                    456k/456k [00:00<00:00, 2.19MB/s]
tokenizer.json: 100%                                1.36M/1.36M [00:00<00:00, 6.26MB/s]
Device set to use cuda:0
```

```
prompt = "Hello, I'm a large language model"
gen(prompt, max_length=40)
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, p
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
[{'generated_text': "Hello, I'm a large language model. Not because I love writing. I
love to write software.\n\nBut if you look carefully at the code I wrote in the
previous entry, there are"}]
```

```
# text2text-generation with encoder-decoder transformer model, performing inferences with pi
text2text_gen = pipeline('text2text-generation',
                         model = 'google/flan-t5-base')
```

```
config.json: 100%                                   1.40k/1.40k [00:00<00:00, 34.8kB/s]
model.safetensors: 100%                             990M/990M [00:05<00:00, 188MB/s]
generation_config.json: 100%                        147/147 [00:00<00:00, 3.88kB/s]
tokenizer_config.json: 100%                         2.54k/2.54k [00:00<00:00, 121kB/s]
spiece.model: 100%                                  792k/792k [00:00<00:00, 25.2MB/s]
tokenizer.json: 100%                                2.42M/2.42M [00:00<00:00, 11.1MB/s]
special_tokens_map.json: 100%                       2.20k/2.20k [00:00<00:00, 101kB/s]
Device set to use cuda:0
```

```
prompt = "Translate from English to French : Hi I am your best buddy since childhood"

text2text_gen(prompt)
```

```
[{'generated_text': 'Hi, je suis votre meilleur ami depuis la naissance'}]
```

```
# let's load llm tiiuae/falcon-7b-instruct and write prompt
model = 'tiiuae/falcon-7b-instruct'
tokenizer = AutoTokenizer.from_pretrained(model)
```

```
device = 'cuda'
```

```
!nvidia-smi
```

```
Mon Feb 10 10:58:03 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. EC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M |
|                                         |                        |               MIG M |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                      |
| N/A   74C    P0             31W /   70W |    1690MiB /  15360MiB |      0%      Defaul |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memor |
|        ID   ID                                                               Usage      |
|=========================================================================================|
+-----------------------------------------------------------------------------------------+
```

```
pipe = pipeline("text-generation",
                model = model,
                tokenizer = tokenizer,
                torch_dtype = torch.bfloat16,
                device=device)
```

```
Loading checkpoint shards: 100%                    2/2 [00:01<00:00,  1.29it/s]
Device set to use cuda
```

We have loaded our llm model until now via text-generation pipeline ok!.

Let's learn to use the prompt for solving many NLP task with our llm model.

1. Text Classification: In text classification task assign a label like positive, negative or neutral to a sequence of text which is also called sentiment analysis.

Let's write a prompt that instructs the model to classify a given text

```
prompt = """Classify the text into neutral, negative or positive.
Text: This movie is definitely one of my favorite movies of its kind.
The interaction between respectable and morally strong characters is an ode to chivalry and
Sentiment:
"""

sequences = pipe(
    prompt,
    max_new_tokens=10,
)

for seq in sequences:
    print(f"Result: {seq['generated_text']}")
```

```
Setting `pad_token_id` to `eos_token_id`:11 for open-end generation.
Result: Classify the text into neutral, negative or positive.
Text: This movie is definitely one of my favorite movies of its kind.
The interaction between respectable and morally strong characters is an ode to chivalry
Sentiment:
Positive
```

2. Named Entity Recognition Named Entity Recognition (NER) is a task of finding named entities in a piece of text, such as a person, location, or organization.

```
prompt = """Return a list of named entities in the text.
Text: The Golden State Warriors are an American professional basketball team based in San Fr
Named entities:
"""

sequences = pipe(
    prompt,
    max_new_tokens=15,
    return_full_text = False,
)

for seq in sequences:
    print(f"{seq['generated_text']}")
```

```
Setting `pad_token_id` to `eos_token_id`:11 for open-end generation.
- Golden State Warriors
- San Francisco
```

3. Translation LLM (encoder-decoder transformer) can also perform the translation from one to another language. We will use Falcon-7b-instruct model and pass a prompt to instruct a model to translate a piece of text from English to Italian. we've added a do_sample=True and top_k=10 to allow the model to be a bit more flexible when generating output.

```
prompt = """Translate the English text to Italian.
Text: Sometimes, I've believed as many as six impossible things before breakfast.
Translation:
"""

sequences = pipe(
    prompt,
    max_new_tokens=20,
    do_sample=True,
    top_k=10,
    return_full_text = False,
)

for seq in sequences:
    print(f"{seq['generated_text']}")
```

```
Setting `pad_token_id` to `eos_token_id`:11 for open-end generation.
Talvolta, ho creduto fino a sei impossibili cose prima di colazione.
```

4. Text Summarization

Text summarization is also an text generative task where output is heavily dependent on the input therefore we will use encoder-decoder transformer model but we can also use the decoder-only transformer for text summarization task also.

Note : We can place the instruction either begining of prompt or end does't matter.

```
prompt = """Permaculture is a design process mimicking the diversity, functionality and resi
Write a summary of the above text.
Summary:
"""

sequences = pipe(
    prompt,
    max_new_tokens=30,
    do_sample=True,
    top_k=10,
    return_full_text = False,
)
```

```
for seq in sequences:
    print(f"{seq['generated_text']}")
```

```
Setting `pad_token_id` to `eos_token_id`:11 for open-end generation.
Permaculture is an ecological design approach mimicking nature's diversity, functionalit
```

5. Question Answering Prompt for question answering task can be structured as follows having logical componenets: instructions, context, questions, leading word/phrase like Answer: to tell the model start generationg answer.

```
prompt = """Answer the question using the context below.
Context: Gazpacho is a cold soup and drink made of raw, blended vegetables. Most gazpacho ir
Question: What modern tool is used to make gazpacho?
Answer:
"""

sequences = pipe(
    prompt,
    max_new_tokens=10,
    do_sample=True,
    top_k=10,
    return_full_text = False,
)

for seq in sequences:
    print(f"Result: {seq['generated_text']}")
```

```
Setting `pad_token_id` to `eos_token_id`:11 for open-end generation.
Result:
A blender or food processor is a modern tool
```

6. Reasoning As human have have build in ability to reason but we have to check does llm able to reason and we found that it's defficult for them to perform the reasoning but if we want to get better reasoning then we have to write the better prompts using prompting techniques like Chain-of-thought.

```
# a model reason about a simple arithmetics task
prompt = """There are 5 groups of students in the class. Each group has 4 students. How many

sequences = pipe(
    prompt,
    max_new_tokens=30,
    do_sample=True,
    top_k=10,
    return_full_text = False,
)
```

```
for seq in sequences:
    print(f"Result: {seq['generated_text']}")
```

```
Setting `pad_token_id` to `eos_token_id`:11 for open-end generation.
Result:
There are 5*4=20 groups in the class. Each group has 4 students. Therefore, there are 20
```

Let's increase the complexity a little and see if we can still get away... It's giving answer 8 which wrong becouse right answer is 12.

```
prompt = """I baked 15 muffins. I ate 2 muffins and gave 5 muffins to a neighbor. My partner

sequences = pipe(
    prompt,
    max_new_tokens=10,
    do_sample=True,
    top_k=10,
    return_full_text = False,
)

for seq in sequences:
    print(f"Result: {seq['generated_text']}")
```

```
Setting `pad_token_id` to `eos_token_id`:11 for open-end generation.
Result:
We are left with 8 muffins.
```

Prompt Engineering Best Practices

1. When choosing the model to work with, the latest and most capable models are likely to perform better.
2. Start with a simple and short prompt, and iterate from there.
3. Put the instructions at the beginning of the prompt, or at the very end. When working with large context, models apply various optimizations to prevent Attention complexity from scaling quadratically. This may make a model more attentive to the beginning or end of a prompt than the middle.
4. Clearly separate instructions from the text they apply to - more on this in the next section.
5. Be specific and descriptive about the task and the desired outcome - its format, length, style, language, etc.
6. Avoid ambiguous descriptions and instructions.
7. Favor instructions that say "what to do" instead of those that say "what not to do".
8. "Lead" the output in the right direction by writing the first word (or even begin the first sentence for the model).

9. Use advanced techniques like Few-shot prompting and Chain-of-thought Test your prompts with different models to assess their robustness.
10. Version and track the performance of your prompts.

Zero-shot prompting model has been given instructions and context but no examples with solutions. Therefore, LLMs that have been fine-tuned on instruction datasets, generally perform well on such “zero-shot” tasks.

However, you may find that your task has more complexity or nuance, and, perhaps, you have some requirements for the output that the model doesn’t catch on just from the instructions. Then Few-shot prompting technique came into existance to tackle this problem ok

Few-shot prompting we provide examples in the prompt giving the model more context to improve the performance. The examples condition the model to generate the output following the patterns in the examples.

Limitations of the few-shot prompting technique:

While LLMs can pick up on the patterns in the examples, these technique doesn’t work well on complex reasoning tasks Few-shot prompting requires creating lengthy prompts. Prompts with large number of tokens can increase computation and latency. There’s also a limit to the length of the prompts. Sometimes when given a number of examples, models can learn patterns that you didn’t intend them to learn, e.g. that the third movie review is always negative.

```
prompt = """Text: The first human went into space and orbited the Earth on April 12, 1961.
Date: 04/12/1961
Text: The first-ever televised presidential debate in the United States took place on Septem
Date:"""

sequences = pipe(
    prompt,
    max_new_tokens=8,
    do_sample=True,
    top_k=10,
)

for seq in sequences:
    print(f"Result: {seq['generated_text']}")
```

```
Setting `pad_token_id` to `eos_token_id`:11 for open-end generation.
Result: Text: The first human went into space and orbited the Earth on April 12, 1961.
Date: 04/12/1961
Text: The first-ever televised presidential debate in the United States took place on Se
Date: 09/28/1960
```

Chain-of-thought Chain-of-thought (CoT) prompting is a technique that nudges a model to produce intermediate reasoning steps thus improving the results on complex reasoning tasks.

There are two ways of steering a model to producing the reasoning steps:

few-shot prompting by illustrating examples with detailed answers to questions, showing the model how to work through a problem. by instructing the model to reason by adding phrases like “Let’s think step by step” or “Take a deep breath and work through the problem step by step.”

```
Start coding or generate with AI.
```

Prompting vs Fine-tuning You can achieve great results by optimizing your prompts, however, you may still ponder whether fine-tuning a model would work better for your case. Here are some scenarios when fine-tuning a smaller model may be a preferred option:

Your domain is wildly different from what LLMs were pre-trained on and extensive prompt optimization did not yield sufficient results. You need your model to work well in a low-resource language. You need the model to be trained on sensitive data that is under strict regulations. You have to use a small model due to cost, privacy, infrastructure or other limitations. In all of the above examples, you will need to make sure that you either already have or can easily obtain a large enough domain-specific dataset at a reasonable cost to fine-tune a model. You will also need to have enough time and resources to fine-tune a model.

Reference

1. LLM Prompting Guide by HuggingFace