Use Case for Generative AI: Dialogue Summarization

Welcome to the hands-on segment of this course. This lab will guide you through the process of dialogue summarization using generative AI. You will investigate the relationship between the input text and the model's output, and engage in prompt engineering to guide the model towards your desired task. By examining zero shot, one shot, and few shot inferences, you will embark on your journey into prompt engineering and observe its potential to improve the generative output of Large Language Models

Content Index

1 - Establish Kernel and Necessary Dependencies

Begin by verifying the selection of the appropriate kernel.

details of the image, kernel, and instance type, simply click on it (located at the top right of the screen)

> Please make sure that you choose **ml.m5.2xlarge** instance type.
> To find that instance type, you might have to scroll down to the "All Instances" section in the dropdown.
> Choice of another instance type might cause training failure/kernel halt/account deactivation.

Next, proceed with the installation of the necessary packages for utilizing PyTorch, Hugging Face transformers, and datasets.

> The next cell may take a few minutes to run. Please be patient.
> Ignore the warnings and errors, along with the note about restarting the kernel at the end.

Entrée [ ]:
```
%pip install --upgrade pip
%pip install --disable-pip-version-check \
    torch==1.13.1 \
    torchdata==0.5.1 --quiet

%pip install \
    transformers==4.27.2 \
    datasets==2.11.0  --quiet
```

Initiate the loading process for the datasets, Large Language Model (LLM), tokenizer, and configurator. If you haven't fully understood these components yet, there's no need to worry; explanations and discussions will be provided later in the notebook.

Entrée [ ]:
```
from datasets import load_dataset
from transformers import AutoModelForSeq2SeqLM
from transformers import AutoTokenizer
from transformers import GenerationConfig
```

2 - Condense Dialogue without Prompt Engineering

For this particular use case, your task involves generating a summary of a dialogue using the pre-trained Large Language Model (LLM) FLAN-T5 from Hugging Face. The comprehensive list of available models within the Hugging Face `transformers` package can be accessed here (https://huggingface.co/docs/transformers/index).

We'll now upload some basic dialogues from the DialogSum (https://huggingface.co/datasets/knkarthick/dialogsum) Hugging Face dataset. This dataset comprises over 10,000 dialogues, each accompanied by manually labeled summaries and topics.

Entrée [ ]:
```
huggingface_dataset_name = "knkarthick/dialogsum"

dataset = load_dataset(huggingface_dataset_name)
```

Print a couple of dialogues with their baseline summaries.

Entrée [ ]:
```
example_indices = [40, 200]

dash_line = '-'.join('' for x in range(100))

for i, index in enumerate(example_indices):
    print(dash_line)
    print('Example ', i + 1)
    print(dash_line)
    print('INPUT DIALOGUE:')
    print(dataset['test'][index]['dialogue'])
    print(dash_line)
    print('BASELINE HUMAN SUMMARY:')
    print(dataset['test'][index]['summary'])
    print(dash_line)
    print()
```

Load the FLAN-T5 model (https://huggingface.co/docs/transformers/model_doc/flan-t5), creating an instance of the `AutoModelForSeq2SeqLM` class with the `.from_pretrained()` method.

Entrée [ ]:
```
model_name='google/flan-t5-base'

model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
```

To engage in encoding and decoding, it's essential to operate with text in a tokenized format. The initial step is **tokenization**, a process involving the division of texts into smaller units suitable for processing by LLM models.

Acquire the FLAN-T5 model's tokenizer by employing the `AutoTokenizer.from_pretrained()` method. The `use_fast` parameter activates the fast tokenizer. Although there's no immediate need for an in-depth exploration of this, you can locate the tokenizer parameters in the documentation (https://huggingface.co/docs/transformers/v4.28.1/en/model_doc/auto#transformers.AutoTok

Entrée [ ]:
```
tokenizer = AutoTokenizer.from_pretrained(model_name, use_fast=True)
```

Evaluate the tokenizer by performing encoding and decoding on a straightforward sentence:

Entrée [ ]:
```
sentence = "What time is it, Tom?"

sentence_encoded = tokenizer(sentence, return_tensors='pt')

sentence_decoded = tokenizer.decode(
        sentence_encoded["input_ids"][0],
        skip_special_tokens=True
    )

print('ENCODED SENTENCE:')
print(sentence_encoded["input_ids"][0])
print('\nDECODED SENTENCE:')
print(sentence_decoded)
```

Now it's time to explore how well the base LLM summarizes a dialogue without any prompt engineering. **Prompt engineering** is an act of a human changing the **prompt** (input) to improve the response for a given task.

Entrée [ ]:
```
for i, index in enumerate(example_indices):
    dialogue = dataset['test'][index]['dialogue']
    summary = dataset['test'][index]['summary']

    inputs = tokenizer(dialogue, return_tensors='pt')
    output = tokenizer.decode(
        model.generate(
            inputs["input_ids"],
            max_new_tokens=50,
        )[0],
        skip_special_tokens=True
    )

    print(dash_line)
    print('Example ', i + 1)
    print(dash_line)
    print(f'INPUT PROMPT:\n{dialogue}')
    print(dash_line)
    print(f'BASELINE HUMAN SUMMARY:\n{summary}')
    print(dash_line)
    print(f'MODEL GENERATION - WITHOUT PROMPT ENGINEERING:\n{output}\n')
```

You can see that the guesses of the model make some sense, but it doesn't seem to be sure what task it is supposed to accomplish. Seems it just makes up the next sentence in the dialogue. Prompt engineering can help here.

3 - Condense Dialogue with an Instruction Prompt

Prompt engineering is an important concept in using foundation models for text generation. You can check out this blog (https://www.amazon.science/blog/emnlp-prompt-engineering-is-the-new-feature-engineering) from Amazon Science for a quick introduction to prompt engineering.

3.1 - Execute Zero Shot Inference with an Instruction Prompt

In order to instruct the model to perform a task - summarize a dialogue - you can take the dialogue and convert it into an instruction prompt. This is often called **zero shot inference**. You can check out this blog from AWS (https://aws.amazon.com/blogs/machine-learning/zero-shot-prompting-for-the-flan-t5-foundation-model-in-amazon-sagemaker-jumpstart/) for a quick description of what zero shot learning is and why it is an important concept to the LLM model.

Wrap the dialogue in a descriptive instruction and see how the generated text will change:

Entrée [ ]:
```
for i, index in enumerate(example_indices):
    dialogue = dataset['test'][index]['dialogue']
    summary = dataset['test'][index]['summary']

    prompt = f"""
Summarize the following conversation.

{dialogue}

Summary:
    """

    # Input constructed prompt instead of the dialogue.
    inputs = tokenizer(prompt, return_tensors='pt')
    output = tokenizer.decode(
        model.generate(
            inputs["input_ids"],
            max_new_tokens=50,
        )[0],
        skip_special_tokens=True
    )

    print(dash_line)
    print('Example ', i + 1)
    print(dash_line)
    print(f'INPUT PROMPT:\n{prompt}')
    print(dash_line)
    print(f'BASELINE HUMAN SUMMARY:\n{summary}')
    print(dash_line)
    print(f'MODEL GENERATION - ZERO SHOT:\n{output}\n')
```

This is much better! But the model still does not pick up on the nuance of the conversations though.

Exercise:

- Experiment with the `prompt` text and see how the inferences will be changed. Will the inferences change if you end the prompt with just empty string vs. `Summary:` ?
- Try to rephrase the beginning of the `prompt` text from `Summarize the following conversation.` to something different - and see how it will influence the generated output.

3.2 - Execute Zero Shot Inference with the Prompt Template from FLAN-T5

Let's use a slightly different prompt. FLAN-T5 has many prompt templates that are published for certain tasks here (https://github.com/google-research/FLAN/tree/main/flan/v2). In the following code, you will use one of the pre-built FLAN-T5 prompts (https://github.com/google-

Entrée [ ]:
```
for i, index in enumerate(example_indices):
    dialogue = dataset['test'][index]['dialogue']
    summary = dataset['test'][index]['summary']

    prompt = f"""
Dialogue:

{dialogue}

What was going on?
"""

    inputs = tokenizer(prompt, return_tensors='pt')
    output = tokenizer.decode(
        model.generate(
            inputs["input_ids"],
            max_new_tokens=50,
        )[0],
        skip_special_tokens=True
    )

    print(dash_line)
    print('Example ', i + 1)
    print(dash_line)
    print(f'INPUT PROMPT:\n{prompt}')
    print(dash_line)
    print(f'BASELINE HUMAN SUMMARY:\n{summary}\n')
    print(dash_line)
    print(f'MODEL GENERATION - ZERO SHOT:\n{output}\n')
```

Notice that this prompt from FLAN-T5 did help a bit, but still struggles to pick up on the nuance of the conversation. This is what you will try to solve with the few shot inferencing.

4 - Condense Dialogue with One Shot and Few Shot Inference

One shot and few shot inference are the practices of providing an LLM with either one or more full examples of prompt-response pairs that match your task - before your actual prompt that you want completed. This is called "in-context learning" and puts your model into a state that understands your specific task. You can read more about it in this blog from HuggingFace (https://huggingface.co/blog/few-shot-learning-gpt-neo-and-inference-api).

4.1 - Execute One Shot Inference

Let's build a function that takes a list of `example_indices_full` , generates a prompt with full examples, then at the end appends the prompt which you want the model to complete

Entrée [ ]:
```
def make_prompt(example_indices_full, example_index_to_summarize):
    prompt = ''
    for index in example_indices_full:
        dialogue = dataset['test'][index]['dialogue']
        summary = dataset['test'][index]['summary']

        # The stop sequence '{summary}\n\n\n' is important for FLAN-T5. Oth
        prompt += f"""
Dialogue:

{dialogue}

What was going on?
{summary}


"""

    dialogue = dataset['test'][example_index_to_summarize]['dialogue']

    prompt += f"""
Dialogue:

{dialogue}

What was going on?
"""

    return prompt
```

Construct the prompt to perform one shot inference:

Entrée [ ]:
```
example_indices_full = [40]
example_index_to_summarize = 200

one_shot_prompt = make_prompt(example_indices_full, example_index_to_summar

print(one_shot_prompt)
```

Now pass this prompt to perform the one shot inference:

Entrée [ ]:
```
summary = dataset['test'][example_index_to_summarize]['summary']

inputs = tokenizer(one_shot_prompt, return_tensors='pt')
output = tokenizer.decode(
    model.generate(
        inputs["input_ids"],
        max_new_tokens=50,
    )[0],
    skip_special_tokens=True
)

print(dash_line)
print(f'BASELINE HUMAN SUMMARY:\n{summary}\n')
print(dash_line)
print(f'MODEL GENERATION - ONE SHOT:\n{output}')
```

4.2 - Execute Few Shot Inference

Let's explore few shot inference by adding two more full dialogue-summary pairs to your prompt.

Entrée [ ]:
```
example_indices_full = [40, 80, 120]
example_index_to_summarize = 200

few_shot_prompt = make_prompt(example_indices_full, example_index_to_summa

print(few_shot_prompt)
```

Now pass this prompt to perform a few shot inference:

Entrée [ ]:
```
summary = dataset['test'][example_index_to_summarize]['summary']

inputs = tokenizer(few_shot_prompt, return_tensors='pt')
output = tokenizer.decode(
    model.generate(
        inputs["input_ids"],
        max_new_tokens=50,
    )[0],
    skip_special_tokens=True
)

print(dash_line)
print(f'BASELINE HUMAN SUMMARY:\n{summary}\n')
print(dash_line)
print(f'MODEL GENERATION - FEW SHOT:\n{output}')
```

In this case, few shot did not provide much of an improvement over one shot inference. And, anything above 5 or 6 shot will typically not help much, either. Also, you need to make sure that you do not exceed the model's input-context length which, in our case, if 512 tokens. Anything above the context length will be ignored.

However, you can see that feeding in at least one full example (one shot) provides the model with more information and qualitatively improves the summary overall.

Exercise:

Experiment with the few shot inferencing.

- Choose different dialogues - change the indices in the `example_indices_full` list and `example_index_to_summarize` value.
- Change the number of shots. Be sure to stay within the model's 512 context length, however.

How well does few shot inferencing work with other examples?

5 - Generative Configuration Parameters for Inference

You can change the configuration parameters of the `generate()` method to see a different output from the LLM. So far the only parameter that you have been setting was `max_new_tokens=50`, which defines the maximum number of tokens to generate. A full list of available parameters can be found in the Hugging Face Generation documentation (https://huggingface.co/docs/transformers/v4.29.1/en/main_classes/text_generation#transfor

A convenient way of organizing the configuration parameters is to use `GenerationConfig` class.

Exercise:

Change the configuration parameters to investigate their influence on the output.

Putting the parameter `do_sample = True`, you activate various decoding strategies which influence the next token from the probability distribution over the entire vocabulary. You can then adjust the outputs changing `temperature` and other parameters (such as `top_k` and `top_p`).

Uncomment the lines in the cell below and rerun the code. Try to analyze the results. You can read some comments below.

Entrée [ ]:
```
generation_config = GenerationConfig(max_new_tokens=50)
# generation_config = GenerationConfig(max_new_tokens=10)
# generation_config = GenerationConfig(max_new_tokens=50, do_sample=True, t
# generation_config = GenerationConfig(max_new_tokens=50, do_sample=True, t
# generation_config = GenerationConfig(max_new_tokens=50, do_sample=True, t

inputs = tokenizer(few_shot_prompt, return_tensors='pt')
output = tokenizer.decode(
    model.generate(
        inputs["input_ids"],
        generation_config=generation_config,
    )[0],
    skip_special_tokens=True
)

print(dash_line)
print(f'MODEL GENERATION - FEW SHOT:\n{output}')
print(dash_line)
print(f'BASELINE HUMAN SUMMARY:\n{summary}\n')
```

Comments related to the choice of the parameters in the code cell above:

- Choosing `max_new_tokens=10` will make the output text too short, so the dialogue summary will be cut.
- Putting `do_sample = True` and changing the temperature value you get more flexibility in the output.

As you can see, prompt engineering can take you a long way for this use case, but there are some limitations. Next, you will start to explore how you can use fine-tuning to help your LLM to understand a particular use case in better depth!

Entrée [ ]:
```

```