Using Prompt Templates

Introduction

In the era of language models, the ability to perform a wide range of tasks is at our fingertips. These models operate on a straightforward principle: they accept a text input sequence and generate an output text sequence. The key factor in this process is the input text or prompt.

Crafting suitable prompts is vital for anyone working with large language models, as poorly constructed prompts yield unsatisfactory outputs, while well-formulated prompts lead to powerful results. Recognizing the importance of prompts, the LangChain library has developed a comprehensive suite of objects tailored for them.

This lesson delves into the nuances of PromptTemplates and how to employ them effectively. A PromptTemplate is a predefined structure or pattern used to construct effective and consistent prompts for large language models. It is a guideline to ensure the input text or prompt is properly formatted.

Here's an example of using a `PromptTemplate` with a single dynamic input for a user query. Remember to define the `OPENAI_API_KEY` in your environment variables with your OPEN AI key. Remember also to install the required packages with the following command: `pip install langchain==0.0.208 deeplake openai tiktoken`.

```
from langchain import LLMChain, PromptTemplate
from langchain.llms import OpenAI


# Before executing the following code, make sure to have
# your OpenAI key saved in the “OPENAI_API_KEY” environment variable.
llm = OpenAI(model_name="text-davinci-003", temperature=0)


template = """Answer the question based on the context below. If the
question cannot be answered using the information provided, answer
with "I don't know".
Context: Quantum computing is an emerging field that leverages quantum mechanics to
solve complex problems faster than classical computers.
...
Question: {query}
Answer: """


prompt_template = PromptTemplate(
    input_variables=["query"],
    template=template
```

```
)


# Create the LLMChain for the prompt
chain = LLMChain(llm=llm, prompt=prompt_template)


# Set the query you want to ask
input_data = {"query": "What is the main advantage of quantum computing over
classical computing?"}


# Run the LLMChain to get the AI-generated answer
response = chain.run(input_data)


print("Question:", input_data["query"])
print("Answer:", response)
```

The sample code.

```
Question: What is the main advantage of quantum computing over classical computing?
Answer:  The main advantage of quantum computing over classical computing is its
ability to solve complex problems faster.
```

The output.
You can edit the `input_data` dictionary with any other question.
The template is a formatted string with a `{query}` placeholder that will be substituted with a real question when applied. To create a PromptTemplate object, two arguments are required:

1. `input_variables`: A list of variable names in the template; in this case, it includes only the query.
2. `template`: The template string containing formatted text and placeholders.

After creating the PromptTemplate object, it can be used to produce prompts with specific questions by providing input data. The input data is a dictionary where the key corresponds to the variable name in the template. The resulting prompt can then be passed to a language model to generate answers.
For more advanced usage, you can create a `FewShotPromptTemplate` with an `ExampleSelector` to select a subset of examples that will be most informative for the language model.

```
from langchain import LLMChain, FewShotPromptTemplate
from langchain.llms import OpenAI


llm = OpenAI(model_name="text-davinci-003", temperature=0)
```

```
examples = [
    {"animal": "lion", "habitat": "savanna"},
    {"animal": "polar bear", "habitat": "Arctic ice"},
    {"animal": "elephant", "habitat": "African grasslands"}
]

example_template = """
Animal: {animal}
Habitat: {habitat}
"""

example_prompt = PromptTemplate(
    input_variables=["animal", "habitat"],
    template=example_template
)

dynamic_prompt = FewShotPromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    prefix="Identify the habitat of the given animal",
    suffix="Animal: {input}\nHabitat:",
    input_variables=["input"],
    example_separator="\n\n",
)

# Create the LLMChain for the dynamic_prompt
chain = LLMChain(llm=llm, prompt=dynamic_prompt)

# Run the LLMChain with input_data
input_data = {"input": "tiger"}
response = chain.run(input_data)

print(response)
```

The sample code.

```
tropical forests and mangrove swamps
```

The output.
Additionally, you can also save your `PromptTemplate` to a file in your local filesystem in JSON or YAML format:

```
prompt_template.save("awesome_prompt.json")
```

And load it back:

```
from langchain.prompts import load_prompt

loaded_prompt = load_prompt("awesome_prompt.json")
```

Let's explore more examples using different types of Prompt Templates. In the next example, we see how to use a few shot prompts to teach the LLM by providing examples to respond sarcastically to questions.

```
from langchain import LLMChain, FewShotPromptTemplate, PromptTemplate
from langchain.llms import OpenAI


llm = OpenAI(model_name="text-davinci-003", temperature=0)


examples = [
    {
        "query": "How do I become a better programmer?",
        "answer": "Try talking to a rubber duck; it works wonders."
    }, {
        "query": "Why is the sky blue?",
        "answer": "It's nature's way of preventing eye strain."
    }
]


example_template = """
User: {query}
AI: {answer}
"""


example_prompt = PromptTemplate(
    input_variables=["query", "answer"],
    template=example_template
```

```
)

prefix = """The following are excerpts from conversations with an AI
assistant. The assistant is typically sarcastic and witty, producing
creative and funny responses to users' questions. Here are some
examples:
"""

suffix = """
User: {query}
AI: """

few_shot_prompt_template = FewShotPromptTemplate(
    examples=examples,
    example_prompt=example_prompt,
    prefix=prefix,
    suffix=suffix,
    input_variables=["query"],
    example_separator="\n\n"
)

# Create the LLMChain for the few_shot_prompt_template
chain = LLMChain(llm=llm, prompt=few_shot_prompt_template)

# Run the LLMChain with input_data
input_data = {"query": "How can I learn quantum computing?"}
response = chain.run(input_data)

print(response)
```

The sample code.

```
Start by studying Schrödinger's cat. That should get you off to a good start.
```

The output.

The `FewShotPromptTemplate` provided in the example demonstrates the power of dynamic prompts. Instead of using a static template, this approach incorporates

examples of previous interactions, allowing the AI better to understand the context and style of the desired response.
Dynamic prompts offer several advantages over static templates:

- **Improved context understanding**: By providing examples, the AI can grasp the context and style of responses more effectively, enabling it to generate responses that are more in line with the desired output.
- **Flexibility**: Dynamic prompts can be easily customized and adapted to specific use cases, allowing developers to experiment with different prompt structures and find the most effective format for their application.
- **Better results**: As a result of the improved context understanding and flexibility, dynamic prompts often yield higher-quality outputs that better match user expectations.

This allows us to take full advantage of the model's capabilities by providing examples and context that guide the AI toward generating more accurate, contextually relevant, and stylistically consistent responses.
Prompt Templates also integrate well with other features in LangChain, like chains, and allow you to control the number of examples included based on query length. This helps in optimizing token usage and managing the balance between the number of examples and prompt size.
To optimize the performance of few-shot learning, providing the model with as many relevant examples as possible without exceeding the maximum context window or causing excessive processing times is crucial. The dynamic inclusion or exclusion of examples allows us to strike a balance between providing sufficient context and maintaining efficiency in the model's operation:

```
examples = [
    {
        "query": "How do you feel today?",
        "answer": "As an AI, I don't have feelings, but I've got jokes!"
    }, {
        "query": "What is the speed of light?",
        "answer": "Fast enough to make a round trip around Earth 7.5 times in one
second!"
    }, {
        "query": "What is a quantum computer?",
        "answer": "A magical box that harnesses the power of subatomic particles to
solve complex problems."
    }, {
        "query": "Who invented the telephone?",
        "answer": "Alexander Graham Bell, the original 'ringmaster'."
    }, {
```

```
        "query": "What programming language is best for AI development?",
        "answer": "Python, because it's the only snake that won't bite."
    }, {
        "query": "What is the capital of France?",
        "answer": "Paris, the city of love and baguettes."
    }, {
        "query": "What is photosynthesis?",
        "answer": "A plant's way of saying 'I'll turn this sunlight into food. You're
welcome, Earth.'"
    }, {
        "query": "What is the tallest mountain on Earth?",
        "answer": "Mount Everest, Earth's most impressive bump."
    }, {
        "query": "What is the most abundant element in the universe?",
        "answer": "Hydrogen, the basic building block of cosmic smoothies."
    }, {
        "query": "What is the largest mammal on Earth?",
        "answer": "The blue whale, the original heavyweight champion of the world."
    }, {
        "query": "What is the fastest land animal?",
        "answer": "The cheetah, the ultimate sprinter of the animal kingdom."
    }, {
        "query": "What is the square root of 144?",
        "answer": "12, the number of eggs you need for a really big omelette."
    }, {
        "query": "What is the average temperature on Mars?",
        "answer": "Cold enough to make a Martian wish for a sweater and a hot cocoa."
    }
]
```

Instead of utilizing the examples list of dictionaries directly, we implement a `LengthBasedExampleSelector` like this:

```
from langchain.prompts.example_selector import LengthBasedExampleSelector


example_selector = LengthBasedExampleSelector(
```

```
    examples=examples,
    example_prompt=example_prompt,
    max_length=100
)
```

By employing the `LengthBasedExampleSelector`, the code dynamically selects and includes examples based on their length, ensuring that the final prompt stays within the desired token limit. The selector is employed to initialize a `dynamic_prompt_template:`

```
dynamic_prompt_template = FewShotPromptTemplate(
    example_selector=example_selector,
    example_prompt=example_prompt,
    prefix=prefix,
    suffix=suffix,
    input_variables=["query"],
    example_separator="\n"
)
```

So, the `dynamic_prompt_template` utilizes the `example_selector` instead of a fixed list of examples. This allows the `FewShotPromptTemplate` to adjust the number of included examples **based on the length of the input query**. By doing so, it optimizes the use of the available context window and ensures that the language model receives an appropriate amount of contextual information.

```
from langchain import LLMChain, FewShotPromptTemplate, PromptTemplate
from langchain.llms import OpenAI
from langchain.prompts.example_selector import LengthBasedExampleSelector

llm = OpenAI(model_name="gpt-3.5-turbo")

# Existing example and prompt definitions, and dynamic_prompt_template initialization

# Create the LLMChain for the dynamic_prompt_template
chain = LLMChain(llm=llm, prompt=dynamic_prompt_template)

# Run the LLMChain with input_data
input_data = {"query": "Who invented the telephone?"}
response = chain.run(input_data)
```

```
print(response)
```

The sample code.

```
Alexander Graham Bell, the man who made it possible to talk to people from miles
away!
```

The output.

Conclusion

Prompt Templates are essential for generating effective prompts for large language models, providing a structured and consistent format that maximizes accuracy and relevance. Integrating dynamic prompts enhances context understanding, flexibility, and results, making them a valuable asset for language model development.

In the next lesson, we'll learn about few shot prompting and example selectors in LangChain.