Chains

Rowel Atienza, PhD
University of the Philippines
github.com/roatienza

2023

Why Chains

- We need an infrastructure to aid the creation of AGI Applications
  - LangChain - https://github.com/hwchase17/langchain
  - Semantic Kernel - https://github.com/microsoft/semantic-kernel

LangChain

```
pip install langchain --upgrade
```

LangChain

Memory

LLMs and Prompts

Agents

Evaluation

Chains

Data Augmented Generation

Large Language Models (LLMs)

- OpenAI

```
from langchain.llms import OpenAI
llm = OpenAI(temperature=0.5)
text = "What are the 10 best places to visit?"
print(llm(text))
```

- LLaMA
- Cohere
- Others

Prompts

- A new way of “programming” LLMs to elicit the desired outcomes

Elements of a Prompt

- Instruction - a specific task or instruction you want the model to perform
- Context - external information or additional context that can steer the model to better responses
- Input Data - the input or question that we are interested to find a response for
- Output Indicator - the type or format of the output.

https://www.promptingguide.ai/

Instruction

- Give a specific task

```
Write
Make a list
Summarize
Plan
```

- Use of separator

```
Human: ### Instruction ###
Translate the text below to Spanish:
Text: "hello!”

AI: Texto: ¡hola!
```

Instruction

- Be specific

  Human: Replace every number by its equivalent in words.

  There are 12 months in one year. In a month, there are 28, 29, 30 or 31 days. A week is made of 7 days.
  A year is made of 365 days.

  AI: "There are twelve months in one year. In a month, there are twenty-eight, twenty-nine, thirty, or thirty-one days. A week is made of seven days. A year is made of three hundred sixty-five days."

Example Prompts

- Summarization
- Information Extraction
- Question Answering
- Text Classification
- Conversion
- Code Generation

```
Human: In bash, find all files /tmp that are older
than 1 week.
AI: find /tmp -type f -mtime +7
```

Prompt Template

```python
from langchain.prompts import PromptTemplate
prompt = PromptTemplate(
     input_variables=["to_do_what"],
     template="What are the 10 best places to {to_do_what}?",
)
formatted_prompt = prompt.format(to_do_what="eat")
print(f"Formatted prompt: {formatted_prompt}")
```

Example Prompts - Reasoning

Human: I have to water my indoor plants everyday. Today, it rained, should I water my plants?

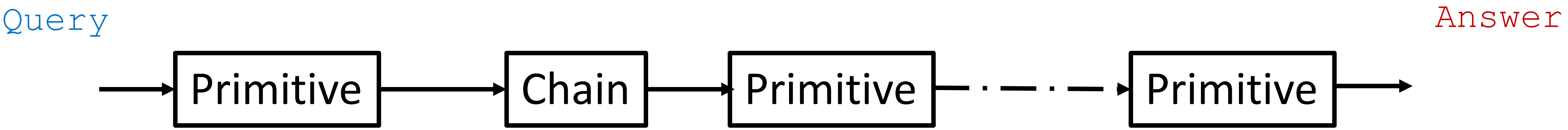
AI: If your indoor plants are not exposed to rain, then the fact that it rained outside should not impact your watering routine for the indoor plants. You should continue to water them as you normally would.

Prompt Engineering

- More on this later…

Chains

- Chains go beyond a single LLM call and involve sequences of calls (whether to an LLM or a different utility).
- LangChain provides a standard interface for chains, lots of integrations with other tools, and end-to-end chains for common applications.



LangChain: Links of Primitives and/or Chains

Chains

```
from langchain.chains import LLMChain

chain = LLMChain(llm=llm, prompt=prompt)
formatted_prompt = prompt.format(to_do_what="...")
query = input(formatted_prompt)
formatted_prompt = prompt.format(to_do_what=query)
print(f"Human: {formatted_prompt}")
results = chain.run(query).split('\n')
print(f"AI: ")
for result in results:
     print(result)
```

Data Augmented Generation

- Data Augmented Generation involves specific types of chains that first interact with an external data source to fetch data for use in the generation step.
- Examples include summarization of long pieces of text and question/answering over specific data sources.

Tool

- A function that performs a specific duty.
  - Google search
  - Database lookup
  - Math solver

```
from langchain.agents import load_tools
tools = load_tools(["serpapi", "llm-math"], llm=llm)
```

Agents

- Agents involve an LLM making decisions about which Actions to take, taking that Action, seeing an Observation, and repeating that until done.
- More on this later…

Memory

- Chains are stateless
- Sometimes, we want to use our previous interactions as context - need for Memory
- Memory refers to persisting state between calls of a chain/agent

Example

```python
from langchain import OpenAI, ConversationChain
from langchain.memory import ConversationBufferMemory

llm = OpenAI(temperature=0.5)
# we use a memory to store the conversation history
memory = ConversationBufferMemory()
conversation = ConversationChain(
     llm=llm,
     verbose=True,
     memory=memory
)
```

Evaluation

- Generative models are notoriously hard to evaluate with traditional metrics.
- Two problems
  - Lack of data
  - Lack of metrics

Code demo

https://github.com/roatienza/Deep-Learning-Experiments/blob/master/versions/2023/llm/python/langchain_quick_demo.ipynb