# 5.1.4 实体记忆和知识图谱记忆

在处理复杂对话时，我们常常需要提取对话中的关键信息。这种需求促使我们发展出了知识图谱和实体记忆这两种主要工具。

知识图谱是一种特殊的记忆类型，它能够根据对话内容构建出一个信息网络。每当它识别到相关的信息，都会接收这些信息并逐步构建出一个小型的知识图谱。与此同时，这种类型的记忆也会产生一种特殊的数据类型——知识图谱数据类型。

另一方面，实体记忆则专注于在对话中提取特定实体的信息。它使用大型语言模型(LLM)提取实体信息，并随着时间的推移，通过同样的方式积累对这个实体的知识。因此，实体记忆的结果通常表现为关于特定事物的关键信息字典。

这两种工具都试图根据对话内容来表示对话，并提取其中的信息。实体记忆和知识图谱记忆就是这种情况的两个例子。

为了让你能更清晰地理解它们的功能，我们在代码示例中放入了一些简单的提示。这些提示能够让你看到当AI仅使用相关信息部分中包含的信息，并且不会产生幻觉时，它们实际上在做什么。我们的目标是让AI更专注于我们所说的内容，而不是无端地添加新的信息。

在实践中，我们通常会使用简单的提示来引导AI的操作。当AI仅使用相关信息部分中包含的信息，并且不会产生幻觉时，这些提示才会被使用。通过这种方式，我们可以确保AI在处理对话时始终保持清晰和准确。

#### 公共代码

先安装库：

!pip -q install openai langchain

设置密钥：

import os  
os.environ['OPENAI\_API\_KEY'] = ''

引入各组件，实例化一个会话链（ConversationChain）。为了让你能更清晰地理解它们的功能，我们在引入了提示模板组件，构造一个提示，方便大家理解。 这里我们使用的 Chain 只是一个简单的对话链 ConversationChain，允许我们跟 OpenAI 模型交互，并传递我们想要说的内容。：

from langchain import OpenAI  
llm = OpenAI(model\_name='text-davinci-003',   
 temperature=0,   
 max\_tokens = 256)  
from langchain.chains import ConversationChain  
from langchain.prompts.prompt import PromptTemplate

#### 知识图谱记忆代码

先导入且实例化组件。

from langchain.chains.conversation.memory import ConversationKGMemory

构建一个简单的提示,让AI仅使用相关信息部分中包含的信息，并且不会产生幻觉。通过这种方式，我们可以确保 AI 在处理对话时始终保持清晰和准确。

template = """The following is a friendly conversation between a human and an AI. The AI is talkative and provides lots of specific details from its context.   
If the AI does not know the answer to a question, it truthfully says it does not know. The AI ONLY uses information contained in the "Relevant Information" section and does not hallucinate.  
  
Relevant Information:  
  
{history}  
  
Conversation:  
Human: {input}  
AI:"""  
prompt = PromptTemplate(  
 input\_variables=["history", "input"], template=template)  
  
conversation = ConversationChain(  
 llm=llm,   
 verbose=True,   
 prompt=prompt,  
 memory=ConversationKGMemory(llm=llm)  
)

让我们开始对话，每次输入后等待AI返回的信息后，再输下一条：

# 请依次运行以下代码，不要一次性运行。  
conversation.predict(input="你好，我叫美丽")  
conversation.predict(input="今天心情怎么样？")  
conversation.predict(input="我想找客户服务中心")  
conversation.predict(input="我的洗衣机坏了,操作面板出现ERROR字样")  
conversation.predict(input="我的保修卡编号是A512423")

执行完最后一条对话后，我们打印记忆组件保存的知识图谱数据的结果：

print(conversation.memory.kg)  
print(conversation.memory.kg.get\_triples())

我们可以看到记忆内存里面保存了对话中的关键信息。

<langchain.graphs.networkx\_graph.NetworkxEntityGraph object at 0x000001C953D48CD0>  
[('AI', 'good mood', 'has a'), ('AI', 'new skills', 'is learning'), ('AI', 'talking to Human', 'enjoys'), ('Customer Service Center', 'city center', 'is located in'), ('Customer Service Center', '24 hour service', 'provides'), ('Customer Service Center', 'website', 'can be found on'), ('Customer Service Center', 'phone', 'can be contacted by'), ('Washing machine', 'ERROR on the control panel', 'has'), ('Human', 'A512423', 'has a warranty card number')]

#### 实体知识记忆

先导入且实例化组件。

from langchain.chains.conversation.memory import ConversationEntityMemory  
from langchain.chains.conversation.prompt import ENTITY\_MEMORY\_CONVERSATION\_TEMPLATE

我们引入Langchain封装好的实体记忆的提示模板 ENTITY\_MEMORY\_CONVERSATION\_TEMPLATE。

conversation = ConversationChain(  
 llm=llm,   
 verbose=True,   
 prompt=ENTITY\_MEMORY\_CONVERSATION\_TEMPLATE,  
 memory=ConversationEntityMemory(llm=llm)  
)

让我们开始对话，每次输入后等待AI返回的信息后，再输下一条：

# 请依次运行以下代码，不要一次性运行。  
conversation.predict(input="你好，我叫美丽")  
conversation.predict(input="今天心情怎么样？")  
conversation.predict(input="我想找客户服务中心")  
conversation.predict(input="我的洗衣机坏了,操作面板出现ERROR字样")  
conversation.predict(input="我的保修卡编号是A512423")

执行完最后一条对话后，我们打印记忆组件保存的知识图谱数据的结果：

print(conversation.memory.entity\_cache )

我们可以看到记忆内存里面保存了对话中的关键信息。

['A512423', 'ERROR']

这两种工具都试图根据对话内容来表示对话，并提取其中的信息。