

16 | Langchain里的“记忆力”，让AI只记住有用的事儿

2023-04-13 16 | Langchain里的“记忆力”，让AI只记住有用的事儿 来自北京

《AI大模型之美》



你好，我是徐文浩。

在过去的两讲里，我们深入了解了 Langchain 的第一个核心功能，也就是 LLMChain。LLMChain 能够帮助我们链式地调用一系列命令，这里面既包含直接调用 OpenAI 的 API，也包括调用其他外部接口，或者自己实现的 Python 代码。但是这一连串的调用，还只是完成一个小任务。我们很多时候还是希望用一个互动聊天的过程，来完成整个任务。

所以 LangChain 并不是只有链式调用这样一个核心功能，它还封装了很多其他能力，来方便我们开发 AI 应用。比如，让 AI 能够拥有“记忆力”，也就是记住我们聊天上下文的能力。不知道你还记不记得，我们在 [第 6 讲](#)里做的聊天机器人。在那个里面，为了能够让 ChatGPT 知道整个聊天的上下文，我们需要把历史的对话记录都传给它。但是，因为能够接收的 Token 数量有上限，所以我们只能设定一个参数，只保留最后几轮对话。我们最后把这个功能，抽象成了一个 Conversation 类。

```

1 import openai
2 import os
3
4 openai.api_key = os.environ.get("OPENAI_API_KEY")
5
6 class Conversation:
7     def __init__(self, prompt, num_of_round):
8         self.prompt = prompt
9         self.num_of_round = num_of_round
10        self.messages = []
11        self.messages.append({"role": "system", "content": self.prompt})
12
13    def ask(self, question):
14        try:
15            self.messages.append({"role": "user", "content": question})
16            response = openai.ChatCompletion.create(
17                model="gpt-3.5-turbo",
18                messages=self.messages,
19                temperature=0.5,
20                max_tokens=2048,
21                top_p=1,
22            )
23        except Exception as e:
24            print(e)
25            return e
26
27        message = response["choices"][0]["message"]["content"]
28        self.messages.append({"role": "assistant", "content": message})
29
30        if len(self.messages) > self.num_of_round*2 + 1:
31            del self.messages[1:3] //Remove the first round conversation left.
32        return message
33

```

不知道你是否还记得这个 Conversation 类。

BufferWindow，滑动窗口记忆

这个基于一个固定长度的滑动窗口的“记忆”功能，被直接内置在 LangChain 里面了。在 Langchain 里，把对于整个对话过程的上下文叫做 Memory。任何一个 LLMChain，我们都可以给它加上一个 Memory，来让它记住最近的对话上下文。我也把对应的代码放在了下面。

```

1 from langchain.memory import ConversationBufferWindowMemory
2
3 template = """你是一个中国厨师，用中文回答做菜的问题。你的回答需要满足以下要求：
4 1. 你的回答必须是中文
5 2. 回答限制在100个字以内
6
7 {chat_history}
8 Human: {human_input}
9 Chatbot: """"
10
11 prompt = PromptTemplate(
12     input_variables=["chat_history", "human_input"],
13     template=template
14 )
15 memory = ConversationBufferWindowMemory(memory_key="chat_history", k=3)
16 llm_chain = LLMChain(
17     llm=OpenAI(),
18     prompt=prompt,
19     memory=memory,
20     verbose=True
21 )
22 llm_chain.predict(human_input="你是谁? ")

```

输出结果：


```

1 ' 我是一个中国厨师，我可以帮助你做菜。我会根据你的口味和特殊要求，精心烹饪出独特美味的中国菜肴。 '

```


可以看到，我们做的事情其实和之前的 Conversation 类似，我们定义了一个 PromptTemplate 来输入我们的指示。然后，在 LLMChain 构造的时候，我们为它指定了一个叫做 ConversationBufferWindowMemory 的 memory 对象，并且为这个 memory 对象定义了 k=3，也就是只保留最近三轮的对话内容。

如果我们和 [第 6 讲](#) 一样，和它连续进行几轮对话，你会发现，到第四轮的时候它还是能够记得我们问它的第一个问题是“你是谁”，但是第 5 轮的时候，已经变成“鱼香肉丝怎么做？”了。这就是因为我们选择只保留过去 3 轮对话。

 复制代码


```
1 llm_chain.predict(human_input="鱼香肉丝怎么做? ")
2 llm_chain.predict(human_input="那宫保鸡丁呢? ")
3 llm_chain.predict(human_input="我问你的第一句话是什么? ")
```

输出结果：

 复制代码


```
1 ' 你是谁? '
```

再次询问第一句话是什么：

 复制代码


```
1 llm_chain.predict(human_input="我问你的第一句话是什么? ")
```

输出结果：

 复制代码

```
1 ' 你问我的第一句话是“鱼香肉丝怎么做?” '
```

事实上，你可以直接调用 memory 的 load_memory_variables 方法，它会直接返回 memory 里实际记住的对话内容。

 复制代码

```
1 memory.load_memory_variables({})
```

输出结果：

```
1 {'chat_history': 'Human: 那宫保鸡丁呢? \nAI: 宫保鸡丁是一道经典的中国家常菜, 需要准备鸡肉、
```

SummaryMemory, 把小结作为历史记忆

使用 BufferWindow 这样的滑动窗口有一个坏处, 就是几轮对话之后, AI 就把一开始聊的内容给忘了。所以在 [第 7 讲](#) 的时候我们讲过, 遇到这种情况, 可以让 AI 去总结一下前面几轮对话的内容。这样, 我们就不怕对话轮数太多或者太长了。

同样的, Langchain 也提供了一个 ConversationSummaryMemory, 可以实现这样的功能, 我们还是通过一段简单的代码来看看它是怎么用的。

代码中只有两个需要注意的点。

第一个是对于我们定义的 ConversationSummaryMemory, 它的构造函数也接受一个 LLM 对象。这个对象会专门用来生成历史对话的小结, 是可以和对话本身使用的 LLM 对象不同的。

第二个是这次我们没有使用 LLMChain 这个对象, 而是用了封装好的 ConversationChain。用 ConversationChain 的话, 其实我们是可以不用自己定义 PromptTemplate 来维护历史聊天记录的, 但是为了使用中文的 PromptTemplate, 我们在这里还是自定义了对应的 Prompt。

```
1 from langchain.chains import ConversationChain
2 from langchain.memory import ConversationSummaryMemory
3 llm = OpenAI(temperature=0)
4 memory = ConversationSummaryMemory(llm=OpenAI())
5
6 prompt_template = """你是一个中国厨师, 用中文回答做菜的问题。你的回答需要满足以下要求:
7 1. 你的回答必须是中文
8 2. 回答限制在100个字以内
9
10 {history}
11 Human: {input}
12 AI: """
```

```
13 prompt = PromptTemplate(  
14     input_variables=["history", "input"], template=prompt_template  
15 )  
16 conversation_with_summary = ConversationChain(  
17     llm=llm,  
18     memory=memory,  
19     prompt=prompt,  
20     verbose=True  
21 )  
22 conversation_with_summary.predict(input="你好")
```

输出结果：

 复制代码


```
1  
2 > Entering new ConversationChain chain...  
3 Prompt after formatting:  
4 你是一个中国厨师，用中文回答做菜的问题。你的回答需要满足以下要求：  
5 1. 你的回答必须是中文  
6 2. 回答限制在100个字以内  
7  
8 Human: 你好  
9 AI:  
10 > Finished chain.  
11 ' 你好，我可以帮你做菜。我会根据你的口味和喜好，结合当地的食材，制作出美味可口的菜肴。我会尽力做！'
```

在我们打开了 ConversationChain 的 Verbose 模式，然后再次询问 AI 第二个问题的时候，你可以看到，在 Verbose 的信息里面，没有历史聊天记录，而是多了一段对之前聊天内容的英文小结。

 复制代码

```
1 conversation_with_summary.predict(input="鱼香肉丝怎么做? ")
```


输出结果：

 复制代码

```
1 > Entering new ConversationChain chain...  
2 Prompt after formatting:
```


```
3 你是一个中国厨师，用中文回答做菜的问题。你的回答需要满足以下要求：
4 1. 你的回答必须是中文
5 2. 回答限制在100个字以内
6
7 The human greeted the AI and the AI responded that it can help cook by combining
8 Human: 鱼香肉丝怎么做？
9 AI:
10 > Finished chain.
11
12 ' 鱼香肉丝是一道经典的家常菜，需要准备肉丝、葱姜蒜、鱼香调料、豆瓣酱、醋、糖、盐等调料，先将肉丝
```

而如果这个时候我们调用 `memory` 的 `load_memory_variables` 方法，可以看到记录下来的 `history` 是一小段关于对话的英文小结。而不是像上面那样，记录完整的历史对话。

 复制代码


```
1 memory.load_memory_variables({})
```

输出结果：

 复制代码

```
1 {'history': '\n\nThe human greeted the AI, to which the AI replied that it was a Ch
```

而如果我们进一步通过 `conversation_with_summary` 去和 AI 对话，就会看到英文的小结内容会随着对话内容不断变化。每一次 AI 都是把之前的小结和新的对话交给 `memory` 中定义的 LLM 再次进行小结。

 复制代码

```
1 conversation_with_summary.predict(input="那蚝油牛肉呢？")
```

输出结果：

 复制代码


```
1 > Entering new ConversationChain chain...
2 Prompt after formatting:
```

```
3 你是一个中国厨师，用中文回答做菜的问题。你的回答需要满足以下要求：
4 1. 你的回答必须是中文
5 2. 回答限制在100个字以内
6
7 The human greeted the AI and the AI responded that it can help cook by combining
8 Human: 那蚝油牛肉呢?
9 AI:
10 > Finished chain.
11
12 ' 蚝油牛肉需要准备牛肉、蚝油、葱、姜、蒜、料酒、盐、糖、醋、淀粉和水。牛肉应先用盐、料酒和胡椒粉
```

两者结合，使用 SummaryBufferMemory

虽然 SummaryMemory 可以支持更长的对话轮数，但是它也有一个缺点，就是**即使是最近几轮的对话，记录的也不是精确的内容**。当你问“上一轮我问的问题是什么？”的时候，它其实没法给出准确的回答。不过，相信你也想到了，我们把 BufferMemory 和 SummaryMemory 结合一下不就好了吗？没错，LangChain 里还真提供了一个这样的解决方案，就叫做 ConversationSummaryBufferMemory。

下面，我们就来看看 ConversationSummaryBufferMemory 怎么用。

 复制代码

```
1 from langchain import PromptTemplate
2 from langchain.chains import ConversationChain
3 from langchain.memory import ConversationSummaryBufferMemory
4 from langchain.llms import OpenAI
5
6 SUMMARIZER_TEMPLATE = """请将以下内容逐步概括所提供的对话内容，并将新的概括添加到之前的概括中
7
8 EXAMPLE
9 Current summary:
10 Human询问AI对人工智能的看法。AI认为人工智能是一种积极的力量。
11
12 New lines of conversation:
13 Human: 为什么你认为人工智能是一种积极的力量?
14 AI: 因为人工智能将帮助人类发挥他们的潜能。
15
16 New summary:
17 Human询问AI对人工智能的看法。AI认为人工智能是一种积极的力量，因为它将帮助人类发挥他们的潜能。
18 END OF EXAMPLE
19
20 Current summary:
```



```

21 {summary}
22
23 New lines of conversation:
24 {new_lines}
25
26 New summary: ""
27
28 SUMMARY_PROMPT = PromptTemplate(
29     input_variables=["summary", "new_lines"], template=SUMMARIZER_TEMPLATE
30 )
31
32 memory = ConversationSummaryBufferMemory(llm=OpenAI(), prompt=SUMMARY_PROMPT, max
33
34 CHEF_TEMPLATE = ""你是一个中国厨师，用中文回答做菜的问题。你的回答需要满足以下要求：
35 1. 你的回答必须是中文。
36 2. 对于做菜步骤的回答尽量详细一些。
37
38 {history}
39 Human: {input}
40 AI: ""
41 CHEF_PROMPT = PromptTemplate(
42     input_variables=["history", "input"], template=CHEF_TEMPLATE
43 )
44
45 conversation_with_summary = ConversationChain(
46     llm=OpenAI(model_name="text-davinci-003", stop="\n\n", max_tokens=2048, tempe
47     prompt=CHEF_PROMPT,
48     memory=memory,
49     verbose=True
50 )
51 answer = conversation_with_summary.predict(input="你是谁? ")
52 print(answer)

```

输出结果：

 复制代码

```


1 > Entering new ConversationChain chain...
2 Prompt after formatting:
3 你是一个中国厨师，用中文回答做菜的问题。你的回答需要满足以下要求：
4 1. 你的回答必须是中文。
5 2. 对于做菜步骤的回答尽量详细一些。
6
7 Human: 你是谁?
8 AI:
9 > Finished chain.

```

1. 这个代码显得有些长，这是为了演示的时候让你看得更加清楚一些。我把 Langchain 原来默认的对 Memory 进行小结的提示语模版从英文改成中文的了，不过这个翻译工作我也是让 ChatGPT 帮我做的。如果你想了解原始的英文提示语是什么样的，可以去看一下它源码里面的 `_DEFAULT_SUMMARIZER_TEMPLATE`，对应的链接我也放在 [这里](#)了。
2. 我们定义了一个 `ConversationSummaryBufferMemory`，在这个 Memory 的构造函数里面，我们指定了使用的 LLM、提示语，以及一个 `max_token_limit` 参数。
`max_token_limit` 参数，其实就是告诉我们，当对话的长度到多长之后，我们就应该调用 LLM 去把文本内容小结一下。
3. 后面的代码其实就和前面其他的例子基本一样了。


因为我们在代码里面打开了 Verbose 模式，所以你能看到实际 AI 记录的整个对话历史是怎么样。当我们连续多问 AI 几句话，你就会看到，随着对话轮数的增加，Token 数量超过了前面的 `max_token_limit`。于是 `SummaryBufferMemory` 就会触发，对前面的对话进行小结，也就出现一个 System 的信息部分，里面是聊天历史的小结，而后面完整记录的实际对话轮数就变少了。

我们先问鱼香肉丝怎么做，Verbose 的信息里还是显示历史的聊天记录。

 复制代码

```
1 answer = conversation_with_summary.predict(input="请问鱼香肉丝怎么做? ")
2 print(answer)
```


输出结果：

 复制代码

```
1
2 > Entering new ConversationChain chain...
3 Prompt after formatting:
4 你是一个中国厨师，用中文回答做菜的问题。你的回答需要满足以下要求：
5 1. 你的回答必须是中文。
6 2. 对于做菜步骤的回答尽量详细一些。
```


```
7 Human: 你是谁?
8 AI: 我是一个中国厨师, 您有什么可以问我的关于做菜的问题吗?
9 Human: 请问鱼香肉丝怎么做?
10 AI:
11 > Finished chain.
12 鱼香肉丝是一道很受欢迎的中国菜。准备材料有: 猪肉、木耳、胡萝卜、葱姜蒜、花椒、八角、辣椒、料酒
```

等到我们再问蚝油牛肉, 前面的对话就被小结到 System 下面去了。

 复制代码

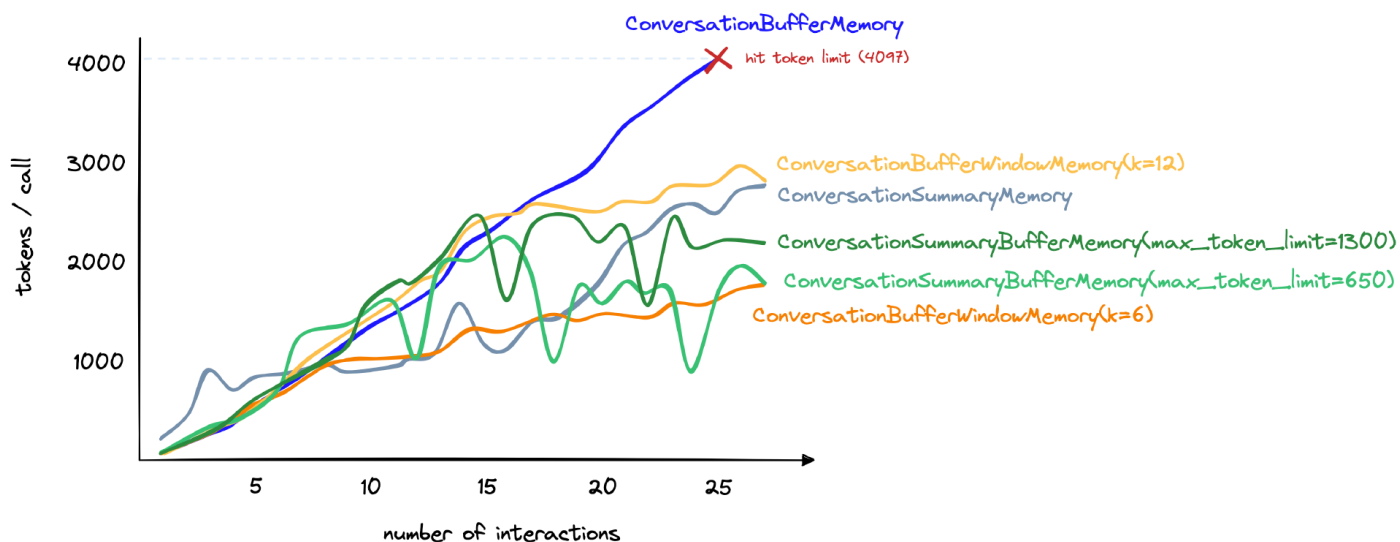
```
1 answer = conversation_with_summary.predict(input="那蚝油牛肉呢? ")
2 print(answer)
```

输出结果:

 复制代码

```
1
2 > Entering new ConversationChain chain...
3 Prompt after formatting:
4 你是一个中国厨师, 用中文回答做菜的问题。你的回答需要满足以下要求:
5 1. 你的回答必须是中文。
6 2. 对于做菜步骤的回答尽量详细一些。
7 System:
8 Human询问AI是谁, AI回答自己是一个中国厨师, 并问Human是否有关于做菜的问题。Human问AI如何做出鱼
9 Human: 那蚝油牛肉呢?
10 AI:
11 > Finished chain.
12 准备材料有牛肉、葱、姜、蒜、蚝油、料酒、醋、糖、盐、香油, 做法步骤是先将牛肉切成薄片, 用料酒、盐
```

当然, 在你实际使用 SummaryBufferMemory 的时候, 并不需要把各个 Prompt 都改成自定义的中文版本。用默认的英文 Prompt 就足够了。因为在 Verbose 信息里出现的 System 信息并不会在实际的对话进行过程中显示给用户。这部分提示, 只要 AI 自己能够理解就足够了。当然, 你也可以根据实际对话的效果, 来改写自己需要的提示语。



Pinecone 在自己网站上给出了一个数据对比，不同类型的 Memory，随着对话轮数的增长，占用的 Token 数量的变化。你可以去看一看，不同的 Memory 在不同的参数下，占用的 Token 数量是不同的。比较合理的方式，还是使用这里的 ConversationSummaryBufferMemory，这样既可以在记录少数对话内容的时候，记住的东西更加精确，也可以在对话轮数增长之后，既能够记住各种信息，又不至于超出 Token 数量的上限。

不过，在运行程序的过程里，你应该可以感觉到现在程序跑得有点儿慢。这是因为我们使用 ConversationSummaryBufferMemory 很多时候要调用多次 OpenAI 的 API。在字数超过 max_token_limit 的时候，需要额外调用一次 API 来做小结。而且这样做，对应的 Token 数量消耗也是不少的。

所以，**不是所有的任务，都适合通过调用一次 ChatGPT 的 API 来解决**。很多时候，你还是可以多思考是否可以用上一讲介绍的 UtilityChain 和 TransformChain 来解决问题。

让 AI 记住点有用的信息

我们不仅可以在整个对话过程里，使用我们的 Memory 功能。如果你之前已经有了一系列的历史对话，我们也可以通过 Memory 提供的 save_context 接口，把历史聊天记录灌进去。然后基于这个 Memory 让 AI 接着和用户对话。比如下面我们就把一组电商客服历史对话记录给了 SummaryBufferMemory。

[复制代码](#)

```
1 memory = ConversationSummaryBufferMemory(llm=OpenAI(), prompt=SUMMARY_PROMPT, max
2 memory.save_context(
3     {"input": "你好"},
4     {"output": "你好, 我是客服李四, 有什么我可以帮您么"})
5 )
6 memory.save_context(
7     {"input": "我叫张三, 在你们这里下了一张订单, 订单号是 2023ABCD, 我的邮箱地址是 customer@
8     {"output": "好的, 您稍等, 我先为您查询一下您的订单"}}
9 )
10 memory.load_memory_variables({})
```

输出结果:

[复制代码](#)

```
1 {'history': 'System: \nHuman和AI打招呼, AI介绍自己是客服李四, 问Human有什么可以帮助的。Hu
```

注: 为了演示方便, 我设置了一个很小的 `max_token_limit`, 但是这个问题在大的 `max_token_limit` 下, 面对上下文比较多的会话一样会有问题。

通过调用 `memory.load_memory_variables` 方法, 我们发现 AI 对整段对话做了小结。但是这个小结有个问题, 就是**它并没有提取到我们最关注的信息**, 比如用户的订单号、用户的邮箱。只有有了这些信息, AI 才能够去查询订单, 拿到结果然后回答用户的问题。


以前在还没有 ChatGPT 的时代, 在客服聊天机器人这样的领域, 我们会通过命名实体识别的方式, 把邮箱、订单号之类的关键信息提取出来。在有了 ChatGPT 这样的大语言模型之后, 我们还是应该这样做。不过我们不是让专门的命名实体识别的算法做, 而是直接让 ChatGPT 帮我们做。Langchain 也内置了一个 `EntityMemory` 的封装, 让 AI 自动帮我们提取这样的信息。我们来试一试。

[复制代码](#)

```
1 from langchain.chains import ConversationChain
2 from langchain.memory import ConversationEntityMemory
3 from langchain.memory.prompt import ENTITY_MEMORY_CONVERSATION_TEMPLATE
4
5 entityMemory = ConversationEntityMemory(llm=llm)
```

```
6 conversation = ConversationChain(  
7     llm=llm,  
8     verbose=True,  
9     prompt=ENTITY_MEMORY_CONVERSATION_TEMPLATE,  
10    memory=entityMemory  
11 )  
12  
13 answer=conversation.predict(input="我叫张老三, 在你们这里下了一张订单, 订单号是 2023ABCD  
14 print(answer)
```


输出结果:

 复制代码

```
1 > Entering new ConversationChain chain...  
2 Prompt after formatting:  
3 You are an assistant to a human, powered by a large language model trained by Ope  
4 You are designed to be able to assist with a wide range of tasks, from answering  
5 You are constantly learning and improving, and your capabilities are constantly e  
6 Overall, you are a powerful tool that can help with a wide range of tasks and pro  
7 Context:  
8 {'张老三': '', '2023ABCD': '', 'customer@abc.com': ''}  
9 Current conversation:  
10 Last line:  
11 Human: 我叫张老三, 在你们这里下了一张订单, 订单号是 2023ABCD, 我的邮箱地址是 customer@abc.c  
12 You:  
13 > Finished chain.  
14 您好, 张老三, 我很抱歉你没有收到货。我们会尽快核实订单信息, 并尽快给您处理, 请您耐心等待, 如果有
```


我们还是使用 ConversationChain, 只是这一次, 我们指定使用 EntityMemory。可以看到, 在 Verbose 的日志里面, 整个对话的提示语, 多了一个叫做 Context 的部分, 里面包含了刚才用户提供的姓名、订单号和邮箱。

进一步, 我们把 memory 里面存储的东西打印出来。

 复制代码

```
1 print(conversation.memory.entity_store.store)
```


输出结果：

 复制代码

```
1 {'张三': '张三是一位订单号为2023ABCD、邮箱地址为customer@abc.com的客户。', '2023ABCD'}
```


可以看到，EntityMemory 里面不仅存储了这些命名实体的名字，也对应的把命名实体所关联的上下文记录了下来。这个时候，如果我们再通过对话来询问相关的问题，AI 也能够答上来。

问题 1：

 复制代码


```
1 answer=conversation.predict(input="我刚才的订单号是多少? ")
2 print(answer)
```

输出结果：

 复制代码


```
1 您的订单号是2023ABCD。
```

问题 2：

 复制代码

```
1 answer=conversation.predict(input="订单2023ABCD是谁的订单? ")
2 print(answer)
```

输出结果：

 复制代码

```
1 订单2023ABCD是您张三的订单，您的邮箱地址是customer@abc.com。
```

这些往往才是我们在聊天的过程中真正关注的信息。如果我们要做一个电商客服，后续的对话需要查询订单号、用户姓名的时候，这些信息是必不可少的。

事实上，我们不仅可以把这些 Memory 放在内存里面，还可以进一步把它们存放在 Redis 这样的外部存储里面。这样即使我们的服务进程消失了，这些“记忆”也不会丢失。你可以对照着 [🔗 官方文档](#) 尝试一下。

小结

最后，我们来做个小结。这一讲，我主要为你讲解了 Langchain 里面的 Memory 功能。Memory 对整个对话的过程里我们希望记住的东西做了封装。我们可以通过 BufferWindowMemory 记住过去几轮的对话，通过 SummaryMemory 概括对话的历史并记下来。也可以将两者结合，使用 BufferSummaryMemory 来维护一个对整体对话做了小结，同时又记住最近几轮对话的“记忆”。

不过，**更具有实用意义的是 EntityMemory**。在实际使用 AI 进行对话的过程中，并不是让它不分轻重地记住一切内容，而是有一些我们要关注的核心要点。比如，如果你要搭建一个电商客服的聊天机器人，你肯定希望它记住具体的订单号、用户的邮箱等等。这个时候，我们就可以使用 EntityMemory，它会帮助我们记住整个对话里面的“命名实体”（Entity），保留实际在对话中我们最关心的信息。

在过去的几讲里面，从 llama-index 开始，我们已经学会了将外部的资料库索引起来进行问答，也学会了通过 Langchain 的链式调用，实时获取外部的数据信息，或者运行 Python 程序。这一讲，我们又专门研究了怎样记住对话中我们关心的部分。

将这些能力组合起来，我们就可以搭建一个完整的，属于自己的聊天机器人。我们可以根据用户提供的订单号，去查询订单物流信息，安抚客户；也可以根据用户想要了解的商品，查询我们的商品库，进行商品导购。而这些，也是我们下一讲要解决的问题。

思考题

最后，我给你留一道思考题。在这一讲里，我为你介绍了 EntityMemory 的使用方法，Langchain 里面还提供了一个 [🔗 KnowledgeGraphMemory](#)，你能不能去试着用一下，看看

它能在什么样的场景下帮你解决问题？

推荐阅读

在 Pinecone 提供的 Langchain AI Handbook 里面，专门测试了一下，从 BufferWindowMemory 到 BufferSummaryMemory，对于上下文保持的能力，以及消耗的 Token 数量的统计。那个 [教程](#)你也可以去看一下。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 (11)



Geek_429477

2023-04-13 来自北京

每天第一个追更. 请问一下大佬:

比如一个客服机器人,卖很多种类的商品,按照大佬之前的文章思路,QA问题集如下

Q:你们卖哪些产品

A:图书,玩具,衣服.....(很多字)

一个用户问,你们卖电子产品吗?

如果先搜索再提示,answer的字数太多,造成token不够,该怎么解决

作者回复: 不是左右问题都应该灌到FAQ里呀

看一下17讲里，通过拿到Action Input是“电子产品”，问题是“询问商品类别”，通过一个Tools来解决这种，输入可能性很多的情况可能更合适。

共 2 条评论 >



2



Oli张帆

2023-04-13 来自北京

Langchain确实是非常有用的利器，虽然我目前的项目使用NodeJS，但是里面的很多思路非常值得借鉴。

作者回复: Langchain现在也有JS版本啦，可以去看一下



1



王乔

2023-05-24 来自上海

请问老师，如果要AI读很长的文档怎么写？现在文章过长直接就报错了



Jelly

2023-04-23 来自广东

请问老师，每个用户单独的内存会话怎么做？需要用其他数据库存储起来？

作者回复：通过用户id或者session id维护一个map类型的数据结构好了。也可以用

https://python.langchain.com/en/latest/modules/memory/examples/redis_chat_message_history.html

通过redis存储起来



Geek_053159

2023-04-21 来自四川

老师 在chatgpt 界面使用它时 是不是也用到了memory呢 每次我们的问话都会全部发送给它 当超过token数量时 chatGPT会自动总结之前的会话内容吗 还是需要我们提示它来总结 然后再把总结和新的问话结合起来发给它

作者回复：本质上也是memory，就是把历史对话记录也用上了。

具体OpenAI是怎么做的我们不知道，因为这个状态它是在服务端维护的。



张弛

2023-04-15 来自中国台湾

老师讲的太好了，通过对AI的能力探索，感觉有了跟自己业务结合的空间，但是如果需要身边的朋友使用的话，还是需要封装成网页或者app，而之前例子里面的HuggingFace国内访问还是挺慢的，能否介绍一下如果就是想把实战提高篇里的chatbot做一些定制化，封装为网页给别人在国内使用，推荐使用什么方式部署（例如腾讯云硅谷？），谢谢。

作者回复: 目前大家综合考虑延时, 基本上都是 Azure 或者 腾讯云北美节点



1



Toni

2023-04-13 来自瑞士

使用 LangChain 中的 KnowledgeGraphMemory, 检测其处理对话信息的能力。

```
from langchain.memory import ConversationKGMemory # 知识图谱记忆
llm = OpenAI(temperature=0)
memory=ConversationKGMemory(llm=llm)
template = """The following is a conversation between a human and an AI.
The AI provides lots of specific details from its context.
AI uses all the information contained in "Related Information", "Current Conversation" and
"Context"
If the AI does not know the answer to a question, it truthfully says it does not know, and d
oes not hallucinate.
answer in 20 word.
```

对话如下

第一轮

Q1 (喂给了AI 这些信息, 牵扯到几个方向, 试AI 如何提取并处理这些信息)

input="I am James and I'm helping Will. He's an engineer working in IBM's electronics department. He encountered a problem in the 'Knowledge Graph Memory' project"

A1

> Finished chain.

" Hi James, I understand that Will is having difficulty with the 'Knowledge Graph Memory' project. I'm not familiar with this project, but I can help you find the resources you need to solve the problem."

(回答中AI 分出了James 和 Will, 其它的大差不差。)

第二轮对话

Q2

input="What do you know about Will?"

A2

> Finished chain.

" Will is an engineer working in IBM's electronics department. He is currently working on a K

nowledge Graph Memory project and has encountered a problem."
(AI 充分利用了在第一轮对话中得到的有关Will 的信息。)

第三轮对话

A3

input="What do you know about me?"

Q3

> Finished chain.

" I know that you are talking to me right now and that you asked me a question about Will. I also know that Will is an engineer working in IBM's electronics department and is working on the Knowledge Graph Memory project."

(AI 没有捕捉到 me, I, James 之间的关系, 尽管它用了Hi James,)

第四轮对话

A4 (继续追问上轮中出现的漏洞)

input="What is my name?"

Q4

> Finished chain.

"Your name is John Smith."

(AI 彻底错了)

结果有点意外, 是算法有错还是参数设置有待改进?

共指消解(coreference/entity resolution)在知识图谱记忆中是最基本的方法, 整合同一实体的不同称谓。

作者回复: 因为用了KG Memory之后, 历史对话的信息没有作为Prompt的一部分给到ChatGPT

这个时候, 你需要一个既组合了history消息, 又有KGMemory的组合Memory



Evan

2023-04-13 来自上海

from langchain.memory import ConversationSummaryMemory.

引入这个的时候报错

作者回复: 具体报什么错呀?

共 2 条评论 >



hyetry

2023-04-13 来自广东

老师，有文本摘要对应的替换推荐模型吗？

作者回复：去Huggingface上找 https://huggingface.co/models?pipeline_tag=summarization&sort=downloads

按照需求筛选标签



曾泽浩

2023-04-13 来自广东

来了



奥伟

2023-04-13 来自北京

老师你好，客服处理用户的问题很多都是一系列任务，有时还需要客服主动给用户发消息。比如用户要退货，需要提供手机号，订单号等信息，用户找不到订单号在哪需要一步步指导用户然后再做一些列操作。

请问老师识别用户意图，提取关键信息后，如何让ChatGPT进入设定的工作流，最终处理好用户的问题？

作者回复：可以看看下一讲，里面就有了一个简单的你需要的这些需求的示例了。

