# 5.1.5 会话摘要记忆

在进行长期的对话时，由于语言模型可接受的令牌范围有限，我们可能无法将所有的对话信息都包含进去。为了解决这个问题，LangChain提供了不同的摘要记忆组件来控制提示的tokens。接下来，我将通过一些代码示例来阐释我们在使用这些不同记忆类型时的区别。

首先，我们需要了解的是，LangChain提供了两种主要的摘要记忆组件：会话缓冲区摘要记忆（ConversationSummaryBufferMemory）和会话摘要记忆（ConversationSummaryMemory）。这两者的定义是什么呢？

会话摘要记忆是一种摘要记忆组件，它不会逐字逐句地存储我们的对话，而是将对话内容进行摘要，并将这些摘要存储起来。这种摘要通常是整个对话的摘要，因此每次我们需要生成新的摘要时，都需要对大型语言模型进行多次调用，以获取响应。

而会话缓冲区摘要记忆则结合了会话摘要记忆的特性和缓冲区的概念。它会保存最近的交互记录，并将旧的交互记录编译成摘要，同时保留两者。但与会话摘要记忆不同的是，会话缓冲区摘要记忆是使用令牌长度而非交互次数来决定何时清除交互记录。这个记忆组件设定了缓冲区的标记数max\_token\_limit，超过此限制的对话记录将被清除。

#### 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）。 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

#### 会话摘要记忆

我们先看会话摘要记忆, 这里先导入且实例化组件。

from langchain.chains.conversation.memory import ConversationSummaryMemory  
memory = ConversationSummaryMemory()

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")

执行完最后一条对话后，我们看到的会话链显示：

'> Entering new chain...  
Prompt after formatting:  
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.  
  
Current conversation:  
  
The human introduces themselves as "美丽", to which the AI responds with a friendly greeting and informs them that they are an AI who can both answer questions and converse. The AI then asks the human what they would like to ask, to which the human responds by asking the AI what their mood is. The AI responds that they are feeling good and are excited to be learning new things, and then asks the human what their mood is. The human then requests to find the customer service center, to which the AI responds that they can help them find it and asks where they want to find the customer service center.  
Human: 我的洗衣机坏了  
AI:  
'> Finished chain.

你会看到，Current conversation 它不会逐字逐句地存储我们的对话，而是将对话内容进行摘要，并将这些摘要存储起来。这种摘要通常是整个对话的摘要。

#### 会话缓冲区摘要记忆

运行完公共代码后，再导入且实例化组件。我们设置为缓冲区的标记数 max\_token\_limit 限制为 40 个标记。

from langchain.chains.conversation.memory import ConversationSummaryBufferMemory  
memory = ConversationSummaryBufferMemory(llm=OpenAI(),max\_token\_limit=40)  
conversation = ConversationChain(  
 llm=llm,   
 verbose=True,   
 memory=memory  
)

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")

执行完最后一条对话后，我们看到的会话链显示：

'> Entering new chain...  
Prompt after formatting:  
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.  
  
Current conversation:  
System:   
  
The AI is introduced and greets the human, telling the human that it is an AI and can answer questions or have a conversation. The AI then asks the human what they would like to ask, to which the human replied asking how the AI is feeling. The AI replied that it was feeling good and was excited for the conversation. The human then asked to find the customer service center, to which the AI replied that it could help the human find the customer service center, and asked if the human wanted to know the location.  
Human: 我的洗衣机坏了  
AI:  
  
'> Finished chain.  
'很抱歉听到你的洗衣机坏了。我可以帮助你找到客户服务中心，你可以联系他们来解决你的问题。你想知道客户服务中心的位置吗？'

你会看到，Current conversation 内容保,丢失了前面对话内容，但做了摘要，将对话以摘要的形式其包含在内，比如丢失了打招呼的对话。在最后给出了范围在40个数量以内的标记的对话记录,即保留了 “Human: 我的洗衣机坏了” 。

但是在这里好像并没有对我们控制提示的长度有明显的改善，只不过是把所有的对话编程了摘要，从结果上来看，似乎比对话更啰嗦，使用的单词数量更多。

#### ##4 更新摘要

我们接着与他对话，不同的是我们抛出更多的问题，之前说洗衣机坏了，现在我抛出一些干扰的对话，比如说手机也坏了。我们看看他的摘要是否会更新？

# 请依次执行而不是一次性执行  
conversation.predict(input="你知道洗衣机的操作屏幕显示ERROR是怎么回事吗?")  
conversation.predict(input="我不知道他们的位置，你可以帮我找到他们的位置吗？")  
conversation.predict(input="我打过他们客服中心的电话了，但是没人接听？")  
conversation.predict(input="我的手机也坏了")

最后我们看看记忆内存中保存了什么：

print(memory.moving\_summary\_buffer)

The AI is introduced and greets the human, telling the human that it is an AI and can answer questions or have a conversation. The AI then asks the human what they would like to ask, to which the human replied asking how the AI is feeling. The AI replied that it was feeling good and was excited for the conversation. The human then asked to find the customer service center, to which the AI replied that it could help the human find the customer service center and asked if the human wanted to know the location. The human then stated that their washing machine and phone were broken, to which the AI apologized and offered to help the human find the customer service center so they can contact them to solve their problem, asking if the human wanted to know the address of the customer service center. The AI then offered to provide the address of the customer service center for the human, asking if they wanted to know it and offering to help the human find other contact methods such as email or social media accounts.

我们可以看到记忆存的是“The human then stated that their washing machine and phone were broken”, 多次对话的摘要，更新了。很早之前我们说洗衣机坏了，但是对话结束说手机坏了。这个组件做摘要记忆的时候，将两个相同的事情合并了。

#### 摘要记忆组件的优势

我们可以看到，会话摘要记忆和会话缓冲区摘要记忆在实现方式上有着显著的差异。由于会话摘要记忆是基于整个对话生成的，所以每次进行新的摘要调用时，我们需要对大型语言模型进行多次调用，以获取对话的摘要。

这两种摘要记忆组件在对话管理中起着重要的作用，特别是在对话的token数超过模型能够处理的范围时，这种摘要能力就显得尤为重要。

无论是对话长度较长，还是需要进行精细管理的情况，会话摘要记忆和会话缓冲区摘要记忆都能够提供有效的帮助。