第三章 储存

在与语言模型交互时，你可能已经注意到一个关键问题：它们并不记忆你之前的交流内容，这在我们构建一些应用程序（如聊天机器人）的时候，带来了很大的挑战，使得对话似乎缺乏真正的连续性。因此，在本节中我们将介绍 LangChain 中的储存模块，即如何将先前的对话嵌入到语言模型中的，使其具有连续对话的能力。

当使用 LangChain 中的储存(Memory)模块时，它旨在保存、组织和跟踪整个对话的历史，从而为用户和模型之间的交互提供连续的上下文。

LangChain 提供了多种储存类型。其中，缓冲区储存允许保留最近的聊天消息，摘要储存则提供了对整个对话的摘要。实体储存则允许在多轮对话中保留有关特定实体的信息。这些记忆组件都是模块化的，可与其他组件组合使用，从而增强机器人的对话管理能力。储存模块可以通过简单的 API 调用来访问和更新，允许开发人员更轻松地实现对话历史记录的管理和维护。

此次课程主要介绍其中四种储存模块，其他模块可查看文档学习。

- 对话缓存储存 (ConversationBufferMemory)
- 对话缓存窗口储存 (ConversationBufferWindowMemory)
- 对话令牌缓存储存 (ConversationTokenBufferMemory)
- 对话摘要缓存储存 (ConversationSummaryBufferMemory)

在 LangChain 中，储存指的是大语言模型（LLM）的短期记忆。为什么是短期记忆？那是因为LLM训练好之后 (获得了一些长期记忆)，它的参数便不会因为用户的输入而发生改变。当用户与训练好的LLM进行对话时，LLM 会暂时记住用户的输入和它已经生成的输出，以便预测之后的输出，而模型输出完毕后，它便会“遗忘”之前用户的输入和它的输出。因此，之前的这些信息只能称作为 LLM 的短期记忆。

为了延长 LLM 短期记忆的保留时间，则需要借助一些外部储存方式来进行记忆，以便在用户与 LLM 对话中，LLM 能够尽可能的知道用户与它所进行的历史对话信息。

一、对话缓存储存

1.1 初始化对话模型

让我们先来初始化对话模型。

```
from langchain.chains import ConversationChain
from langchain.chat_models import ChatOpenAI
from langchain.memory import ConversationBufferMemory

# 这里我们将参数temperature设置为0.0，从而减少生成答案的随机性。
# 如果你想要每次得到不一样的有新意的答案，可以尝试增大该参数。
llm = ChatOpenAI(temperature=0.0)
memory = ConversationBufferMemory()


# 新建一个 ConversationChain Class 实例
# verbose参数设置为True时，程序会输出更详细的信息，以提供更多的调试或运行时信息。
# 相反，当将verbose参数设置为False时，程序会以更简洁的方式运行，只输出关键的信息。
conversation = ConversationChain(llm=llm, memory = memory, verbose=True )
```

1.2 第一轮对话

当我们运行预测(predict)时，生成了一些提示，如下所见，他说“以下是人类和 AI 之间友好的对话，AI 健谈”等等，这实际上是 LangChain 生成的提示，以使系统进行希望和友好的对话，并且必须保存对话，并提示了当前已完成的模型链。

```
conversation.predict(input="你好，我叫皮皮鲁")
```

```
> 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:

Human: 你好，我叫皮皮鲁
AI:

> Finished chain.
```

```
'你好，皮皮鲁！很高兴认识你。我是一个AI助手，可以回答你的问题和提供帮助。有什么我可以帮你的吗？'
```

1.3 第二轮对话

当我们进行第二轮对话时，它会保留上面的提示

```
conversation.predict(input="1+1等于多少？")
```

```
> Entering new ConversationChain 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:
Human: 你好，我叫皮皮鲁
AI: 你好，皮皮鲁！很高兴认识你。我是一个AI助手，可以回答你的问题和提供帮助。有什么我可以帮你的吗？
Human: 1+1等于多少？
AI:

> Finished chain.
```

```
'1+1等于2。'
```

1.4 第三轮对话

为了验证他是否记忆了前面的对话内容，我们让他回答前面已经说过的内容（我的名字），可以看到他确实输出了正确的名字，因此这个对话链随着往下进行会越来越长。

```
conversation.predict(input="我叫什么名字？")
```

```
> Entering new ConversationChain 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:
Human: 你好，我叫皮皮鲁
AI: 你好，皮皮鲁！很高兴认识你。我是一个AI助手，可以回答你的问题和提供帮助。有什么我可以帮你的吗？
Human: 1+1等于多少？
AI: 1+1等于2。
Human: 我叫什么名字？
AI:

> Finished chain.
```

```
'你叫皮皮鲁。'
```

1.5查看储存缓存

储存缓存(buffer)，即储存了当前为止所有的对话信息

```
print(memory.buffer)
```

```
Human: 你好，我叫皮皮鲁
AI: 你好，皮皮鲁！很高兴认识你。我是一个AI助手，可以回答你的问题和提供帮助。有什么我可以帮你的吗？
Human: 1+1等于多少？
AI: 1+1等于2。
Human: 我叫什么名字？
AI: 你叫皮皮鲁。
```

也可以通过`load_memory_variables({})`打印缓存中的历史消息。这里的`{}`是一个空字典，有一些更高级的功能，使用户可以使用更复杂的输入，具体可以通过 LangChain 的官方文档查询更高级的用法。

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

```
{'history': 'Human: 你好，我叫皮皮鲁\nAI: 你好，皮皮鲁！很高兴认识你。我是一个AI助手，可以回答你的问题和提供帮助。有什么我可以帮你的吗？\nHuman: 1+1等于多少？\nAI: 1+1等于2。\nHuman: 我叫什么名字？\nAI: 你叫皮皮鲁。'}
```

1.6 直接添加内容到储存缓存

我们可以使用`save_context`来直接添加内容到`buffer`中。

```
memory = ConversationBufferMemory()
memory.save_context({"input": "你好，我叫皮皮鲁"}, {"output": "你好啊，我叫鲁西西"})
memory.load_memory_variables({})
```

```
{'history': 'Human: 你好，我叫皮皮鲁\nAI: 你好啊，我叫鲁西西'}
```

继续添加新的内容

```
memory.save_context({"input": "很高兴和你成为朋友！"}, {"output": "是的，让我们一起去冒险吧！"})
memory.load_memory_variables({})
```

```
{'history': 'Human: 你好，我叫皮皮鲁\nAI: 你好啊，我叫鲁西西\nHuman: 很高兴和你成为朋友！\nAI: 是的，让我们一起去冒险吧！'}
```

可以看到对话历史都保存下来了！

当我们在使用大型语言模型进行聊天对话时，**大型语言模型本身实际上是无状态的。语言模型本身并不记得到目前为止的历史对话。**每次调用API结点都是独立的。储存(Memory)可以储存到目前为止的所有术语或对话，并将其输入或附加上下文到LLM中用于生成输出。如此看起来就好像它在进行下一轮对话的时候，记得之前说过什么。

二、对话缓存窗口储存

随着对话变得越来越长，所需的内存量也变得非常长。将大量的tokens发送到LLM的成本，也会变得更加昂贵，这也就是为什么API的调用费用，通常是基于它需要处理的tokens数量而收费的。

针对以上问题，LangChain也提供了几种方便的储存方式来保存历史对话。其中，对话缓存窗口储存只保留一个窗口大小的对话。它只使用最近的n次交互。这可以用于保持最近交互的滑动窗口，以便缓冲区不会过大。

2.1 添加两轮对话到窗口储存

我们先来尝试一下使用`ConversationBufferWindowMemory`来实现交互的滑动窗口，并设置`k=1`，表示只保留一个对话记忆。接下来我们手动添加两轮对话到窗口储存中，然后查看储存的对话。

```
from langchain.memory import ConversationBufferWindowMemory

# k=1表明只保留一个对话记忆
memory = ConversationBufferWindowMemory(k=1)
memory.save_context({"input": "你好，我叫皮皮鲁"}, {"output": "你好啊，我叫鲁西西"})
memory.save_context({"input": "很高兴和你成为朋友！"}, {"output": "是的，让我们一起去冒险吧！"})
memory.load_memory_variables({})
```

```
{'history': 'Human: 很高兴和你成为朋友！\nAI: 是的，让我们一起去冒险吧！'}
```

通过结果，我们可以看到窗口储存中只有最后一轮的聊天记录。

2.2 在对话链中应用窗口储存

接下来，让我们来看看如何在`ConversationChain`中运用`ConversationBufferWindowMemory`吧！

```
llm = ChatOpenAI(temperature=0.0)
memory = ConversationBufferWindowMemory(k=1)
conversation = ConversationChain(llm=llm, memory=memory, verbose=False  )

print("第一轮对话: ")
print(conversation.predict(input="你好, 我叫皮皮鲁"))

print("第二轮对话: ")
print(conversation.predict(input="1+1等于多少？"))

print("第三轮对话: ")
print(conversation.predict(input="我叫什么名字？"))
```

```
第一轮对话:
你好，皮皮鲁！很高兴认识你。我是一个AI助手，可以回答你的问题和提供帮助。有什么我可以帮你的吗？
第二轮对话:
1+1等于2。
第三轮对话:
很抱歉，我无法知道您的名字。
```

注意此处！由于这里用的是一个窗口的记忆，因此只能保存一轮的历史消息，因此AI并不能知道你第一轮对话中提到的名字，他最多只能记住上一轮（第二轮）的对话信息

三、对话字符缓存储存

使用对话字符缓存记忆，内存将限制保存的token数量。如果字符数量超出指定数目，它会切掉这个对话的早期部分
以保留与最近的交流相对应的字符数量，但不超过字符限制。

添加对话到Token缓存储存,限制token数量，进行测试

```
from langchain.llms import OpenAI
from langchain.memory import ConversationTokenBufferMemory
memory = ConversationTokenBufferMemory(llm=llm, max_token_limit=30)
memory.save_context({"input": "朝辞白帝彩云间，"}, {"output": "千里江陵一日还。"})
memory.save_context({"input": "两岸猿声啼不住，"}, {"output": "轻舟已过万重山。"})
memory.load_memory_variables({})
```

```
{'history': 'AI: 轻舟已过万重山。'}
```

ChatGPT 使用一种基于字节对编码（Byte Pair Encoding，BPE）的方法来进行 tokenization （将输入文本拆分为token）。BPE 是一种常见的 tokenization 技术，它将输入文本分割成较小的子词单元。OpenAI 在其官方 GitHub 上公开了一个最新的开源 Python 库 tiktoken(https://github.com/openai/tiktoken)，这个库主要是用来计算 tokens 数量的。相比较 HuggingFace 的 tokenizer ，其速度提升了好几倍。

具体 token 计算方式,特别是汉字和英文单词的 token 区别，具体可参考知乎文章(https://www.zhihu.com/question/594159910)。

四、对话摘要缓存储存

对话摘要缓存储存，**使用 LLM 对到目前为止历史对话自动总结摘要**，并将其保存下来。

4.1 使用对话摘要缓存储存

我们创建了一个长字符串，其中包含某人的日程安排。

```
from langchain.chains import ConversationChain
from langchain.chat_models import ChatOpenAI
from langchain.memory import ConversationSummaryBufferMemory

# 创建一个长字符串
schedule = "在八点你和你的产品团队有一个会议。 \
你需要做一个PPT。 \
上午9点到12点你需要忙于LangChain。\
Langchain是一个有用的工具，因此你的项目进展的非常快。\
中午，在意大利餐厅与一位开车来的顾客共进午餐 \
走了一个多小时的路程与你见面，只为了解最新的 AI。 \
确保你带了笔记本电脑可以展示最新的 LLM 样例."

llm = ChatOpenAI(temperature=0.0)
memory = ConversationSummaryBufferMemory(llm=llm, max_token_limit=100)
memory.save_context({"input": "你好，我叫皮皮鲁"}, {"output": "你好啊，我叫鲁西西"})
memory.save_context({"input": "很高兴和你成为朋友！"}, {"output": "是的，让我们一起去冒险吧！"})
memory.save_context({"input": "今天的日程安排是什么？"}, {"output": f"{schedule}"})

print(memory.load_memory_variables({})['history'])
```

```
System: The human introduces themselves as Pipilu and the AI introduces
themselves as Luxixi. They express happiness at becoming friends and decide to go
on an adventure together. The human asks about the schedule for the day. The AI
informs them that they have a meeting with their product team at 8 o'clock and
need to prepare a PowerPoint presentation. From 9 am to 12 pm, they will be busy
with LangChain, a useful tool that helps their project progress quickly. At noon,
they will have lunch with a customer who has driven for over an hour just to
learn about the latest AI. The AI advises the human to bring their laptop to
showcase the latest LLM samples.
```

4.2 基于对话摘要缓存储存的对话链

基于上面的对话摘要缓存储存，我们新建一个对话链。

```
conversation = ConversationChain(llm=llm, memory=memory, verbose=True)
conversation.predict(input="展示什么样的样例最好呢？")
```

```
> Entering new ConversationChain 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 human introduces themselves as Pipilu and the AI introduces themselves as Luxixi. They express happiness at becoming friends and decide to go on an adventure together. The human asks about the schedule for the day. The AI informs them that they have a meeting with their product team at 8 o'clock and need to prepare a PowerPoint presentation. From 9 am to 12 pm, they will be busy with LangChain, a useful tool that helps their project progress quickly. At noon, they will have lunch with a customer who has driven for over an hour just to learn about the latest AI. The AI advises the human to bring their laptop to showcase the latest LLM samples.
Human: 展示什么样的样例最好呢？
AI:

> Finished chain.
```

```
'展示一些具有多样性和创新性的样例可能是最好的选择。你可以展示一些不同领域的应用，比如自然语言处理、图像识别、语音合成等。另外，你也可以展示一些具有实际应用价值的样例，比如智能客服、智能推荐等。总之，选择那些能够展示出我们AI技术的强大和多样性的样例会给客户留下深刻的印象。'
```

```
print(memory.load_memory_variables({}))  # 摘要记录更新了
```

```
{'history': "System: The human introduces themselves as Pipilu and the AI introduces themselves as Luxixi. They express happiness at becoming friends and decide to go on an adventure together. The human asks about the schedule for the day. The AI informs them that they have a meeting with their product team at 8 o'clock and need to prepare a PowerPoint presentation. From 9 am to 12 pm, they will be busy with LangChain, a useful tool that helps their project progress quickly. At noon, they will have lunch with a customer who has driven for over an hour just to learn about the latest AI. The AI advises the human to bring their laptop to showcase the latest LLM samples. The human asks what kind of samples would be best to showcase. The AI suggests that showcasing diverse and innovative samples would be the best choice. They recommend demonstrating applications in different fields such as natural language processing, image recognition, and speech synthesis. Additionally, they suggest showcasing practical examples like intelligent customer service and personalized recommendations to impress the customer with the power and versatility of their AI technology."}
```

通过对比上一次输出，发现摘要记录更新了，添加了最新一次对话的内容总结。

英文版提示

1.对话缓存储存

```
from langchain.chains import ConversationChain
from langchain.chat_models import ChatOpenAI
from langchain.memory import ConversationBufferMemory
```

```python
llm = ChatOpenAI(temperature=0.0)
memory = ConversationBufferMemory()
conversation = ConversationChain(llm=llm, memory = memory, verbose=True )

print("第一轮对话: ")
conversation.predict(input="Hi, my name is Andrew")

print("第二轮对话: ")
conversation.predict(input="What is 1+1?")

print("第三轮对话: ")
conversation.predict(input="What is my name?")
```

```
第一轮对话:
> 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:

Human: Hi, my name is Andrew
AI:

> Finished chain.

第二轮对话:
> 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:
Human: Hi, my name is Andrew
AI: Hello Andrew! It's nice to meet you. How can I assist you today?
Human: What is 1+1?
AI:

> Finished chain.

第三轮对话:
> 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:
Human: Hi, my name is Andrew
AI: Hello Andrew! It's nice to meet you. How can I assist you today?
Human: What is 1+1?
```

```
AI: 1+1 is equal to 2.
Human: What is my name?
AI:

> Finished chain.
```

```
'Your name is Andrew.'
```

```
print("查看储存缓存方式一：")
print(memory.buffer)

print("查看储存缓存方式二：")
print(memory.load_memory_variables({}))

print("向缓存区添加指定对话的输入输出，并查看")
memory = ConversationBufferMemory()  # 新建一个空的对话缓存记忆
memory.save_context({"input": "Hi"}, {"output": "What's up"})  # 向缓存区添加指定对话的输入输出
print(memory.buffer)  # 查看缓存区结果
print(memory.load_memory_variables({}))# 再次加载记忆变量

print("继续向向缓存区添加指定对话的输入输出，并查看")
memory.save_context({"input": "Not much, just hanging"}, {"output": "Cool"})
print(memory.buffer)  # 查看缓存区结果
print(memory.load_memory_variables({}))# 再次加载记忆变量
```

```
查看储存缓存方式一：
Human: Hi, my name is Andrew
AI: Hello Andrew! It's nice to meet you. How can I assist you today?
Human: What is 1+1?
AI: 1+1 is equal to 2.
Human: What is my name?
AI: Your name is Andrew.
查看储存缓存方式二：
{'history': "Human: Hi, my name is Andrew\nAI: Hello Andrew! It's nice to meet you. How can I assist you today?\nHuman: What is 1+1?\nAI: 1+1 is equal to 2.\nHuman: What is my name?\nAI: Your name is Andrew."}
向缓存区添加指定对话的输入输出，并查看
Human: Hi
AI: What's up
{'history': "Human: Hi\nAI: What's up"}
继续向向缓存区添加指定对话的输入输出，并查看
Human: Hi
AI: What's up
Human: Not much, just hanging
AI: Cool
{'history': "Human: Hi\nAI: What's up\nHuman: Not much, just hanging\nAI: Cool"}
```

2. 对话缓存窗口储存

```
from langchain.memory import ConversationBufferWindowMemory
```

```
# k 为窗口参数，k=1表明只保留一个对话记忆
memory = ConversationBufferWindowMemory(k=1)

# 向memory添加两轮对话
memory.save_context({"input": "Hi"}, {"output": "What's up"})
memory.save_context({"input": "Not much, just hanging"}, {"output": "Cool"})

# 并查看记忆变量当前的记录
memory.load_memory_variables({})


llm = ChatOpenAI(temperature=0.0)
memory = ConversationBufferWindowMemory(k=1)
conversation = ConversationChain(llm=llm, memory=memory, verbose=False  )


print("第一轮对话: ")
print(conversation.predict(input="Hi, my name is Andrew"))

print("第二轮对话: ")
print(conversation.predict(input="What is 1+1?"))

print("第三轮对话: ")
print(conversation.predict(input="What is my name?"))
```

```
第一轮对话:
Hello Andrew! It's nice to meet you. How can I assist you today?
第二轮对话:
1+1 is equal to 2.
第三轮对话:
I'm sorry, but I don't have access to personal information.
```

3. 对话字符缓存储存

```
from langchain.llms import OpenAI
from langchain.memory import ConversationTokenBufferMemory
memory = ConversationTokenBufferMemory(llm=llm, max_token_limit=30)
memory.save_context({"input": "AI is what?!"}, {"output": "Amazing!"})
memory.save_context({"input": "Backpropagation is what?"}, {"output":
"Beautiful!"})
memory.save_context({"input": "Chatbots are what?"}, {"output": "Charming!"})
print(memory.load_memory_variables({}))
```

```
{'history': 'AI: Beautiful!\nHuman: Chatbots are what?\nAI: Charming!'}
```

4. 对话摘要缓存储存

```
from langchain.chains import ConversationChain
from langchain.chat_models import ChatOpenAI
from langchain.memory import ConversationSummaryBufferMemory

# 创建一个长字符串
schedule = "There is a meeting at 8am with your product team. \
```

```
You will need your powerpoint presentation prepared. \
9am-12pm have time to work on your LangChain \
project which will go quickly because Langchain is such a powerful tool. \
At Noon, lunch at the italian resturant with a customer who is driving \
from over an hour away to meet you to understand the latest in AI. \
Be sure to bring your laptop to show the latest LLM demo."

# 使用对话摘要缓存
llm = ChatOpenAI(temperature=0.0)
memory = ConversationSummaryBufferMemory(llm=llm, max_token_limit=100)
memory.save_context({"input": "Hello"}, {"output": "What's up"})
memory.save_context({"input": "Not much, just hanging"}, {"output": "Cool"})
memory.save_context({"input": "What is on the schedule today?"}, {"output": f"
{schedule}"})

print("查看对话摘要缓存储存")
print(memory.load_memory_variables({})['history'])

conversation = ConversationChain(llm=llm, memory=memory, verbose=True)

print("基于对话摘要缓存储存的对话链")
conversation.predict(input="What would be a good demo to show?")

print("再次查看对话摘要缓存储存")
print(memory.load_memory_variables({})['history'])
```

```
查看对话摘要缓存储存
System: The human and AI exchange greetings. The human asks about the schedule
for the day. The AI provides a detailed schedule, including a meeting with the
product team, work on the LangChain project, and a lunch meeting with a customer
interested in AI. The AI emphasizes the importance of bringing a laptop to
showcase the latest LLM demo during the lunch meeting.

基于对话摘要缓存储存的对话链
> 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 human and AI exchange greetings. The human asks about the schedule
for the day. The AI provides a detailed schedule, including a meeting with the
product team, work on the LangChain project, and a lunch meeting with a customer
interested in AI. The AI emphasizes the importance of bringing a laptop to
showcase the latest LLM demo during the lunch meeting.
Human: What would be a good demo to show?
AI:

> Finished chain.

再次查看对话摘要缓存储存
```

```
System: The human and AI exchange greetings and discuss the schedule for the day. The AI provides a detailed schedule, including a meeting with the product team, work on the LangChain project, and a lunch meeting with a customer interested in AI. The AI emphasizes the importance of bringing a laptop to showcase the latest LLM demo during the lunch meeting. The human asks what would be a good demo to show, and the AI suggests showcasing the latest LLM (Language Model) demo. The LLM is a cutting-edge AI model that can generate human-like text based on a given prompt. It has been trained on a vast amount of data and can generate coherent and contextually relevant responses. By showcasing the LLM demo, the AI can demonstrate the capabilities of their AI technology and how it can be applied to various industries and use cases.
```