Conversation summary memory

Now let's take a look at using a slightly more complex type of memory - `ConversationSummaryMemory`. This type of memory creates a summary of the conversation over time. This can be useful for condensing information from the conversation over time. Conversation summary memory summarizes the conversation as it happens and stores the current summary in memory. This memory can then be used to inject the summary of the conversation so far into a prompt/chain. This memory is most useful for longer conversations, where keeping the past message history in the prompt verbatim would take up too many tokens.

Let's first explore the basic functionality of this type of memory.

```
from langchain.memory import ConversationSummaryMemory, ChatMessageHistory
from langchain.llms import OpenAI
```

```
memory = ConversationSummaryMemory(llm=OpenAI(temperature=0))
memory.save_context({"input": "hi"}, {"output": "whats up"})
```

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

```
    {'history': '\nThe human greets the AI, to which the AI responds.'}
```

We can also get the history as a list of messages (this is useful if you are using this with a chat model).

```
memory = ConversationSummaryMemory(llm=OpenAI(temperature=0), return_messages=True)
memory.save_context({"input": "hi"}, {"output": "whats up"})
```

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

```
    {'history': [SystemMessage(content='\nThe human greets the AI, to which the AI responds.',
additional_kwargs={})]}
```

We can also utilize the `predict_new_summary` method directly.

```
messages = memory.chat_memory.messages
previous_summary = ""
memory.predict_new_summary(messages, previous_summary)
```

```
    '\nThe human greets the AI, to which the AI responds.'
```

Initializing with messages

If you have messages outside this class, you can easily initialize the class with ChatMessageHistory. During loading, a summary will be calculated.

```
history = ChatMessageHistory()
history.add_user_message("hi")
```

```
history.add_ai_message("hi there!")
```

```
memory = ConversationSummaryMemory.from_messages(llm=OpenAI(temperature=0), chat_memory=history,
return_messages=True)
```

```
memory.buffer
```

```
    '\nThe human greets the AI, to which the AI responds with a friendly greeting.'
```

Using in a chain

Let's walk through an example of using this in a chain, again setting `verbose=True` so we can see the prompt.

```
from langchain.llms import OpenAI
from langchain.chains import ConversationChain
llm = OpenAI(temperature=0)
conversation_with_summary = ConversationChain(
    llm=llm,
    memory=ConversationSummaryMemory(llm=OpenAI()),
    verbose=True
)
conversation_with_summary.predict(input="Hi, what's up?")
```

```
    > 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: Hi, what's up?
    AI:

    > Finished chain.
```

```
    " Hi there! I'm doing great. I'm currently helping a customer with a technical issue. How about you?"
```

```
conversation_with_summary.predict(input="Tell me more about it!")
```

```
    > 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:

    The human greeted the AI and asked how it was doing. The AI replied that it was doing great and was currently helping a customer with a technical issue.
    Human: Tell me more about it!
    AI:

    > Finished chain.
```

```
    " Sure! The customer is having trouble with their computer not connecting to the internet. I'm helping them troubleshoot the issue and figure out what the problem is. So far, we've tried resetting the router and checking the network settings, but the issue still persists. We're currently looking into other possible solutions."
```

```
conversation_with_summary.predict(input="Very cool -- what is the scope of the project?")
```

```
    > 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:

    The human greeted the AI and asked how it was doing. The AI replied that it was doing great and was
```

```
currently helping a customer with a technical issue where their computer was not connecting to the internet.
The AI was troubleshooting the issue and had already tried resetting the router and checking the network
settings, but the issue still persisted and they were looking into other possible solutions.
    Human: Very cool -- what is the scope of the project?
    AI:

    > Finished chain.



    " The scope of the project is to troubleshoot the customer's computer issue and find a solution that will
allow them to connect to the internet. We are currently exploring different possibilities and have already
tried resetting the router and checking the network settings, but the issue still persists."
```