Memory-based agent

正文开始之前先放一个吴恩达老师对Agent的理解图，其中的很多方法在前面的笔记中已经介绍过了：

- **任务分解**是指借助LLM将任务拆解为若干个子任务，并依次对每个子任务进行规划。可以分为先分解后规划（HuggingGPT，plan and solve）和边分解边规划（COT，react，pal）两个思路
- **多方案选择**是指大型语言模型深入“思考”，针对特定任务提出多种可能的方案。接着，利用针对性的任务搜索机制，从中挑选一个最合适的方案来实施。例如ToT，GoT，LAT
- **外部模块辅助规划**。该策略专门设计用于引入外部规划器，以增强规划过程的效率和计划的可行性，同时大型语言模型主要负责将任务规范化。分为符号规划器（LLM+P）和神经规划器（利用强化学习训练深度学习模型，作为决策模型）
- **反思与优化**。这种策略着重于通过自我反思和细节完善来增强规划能力。它激励大型语言模型Agent应用在遭遇失败后进行深入反思，并据此优化规划方案。例如Reflexion，critic
- **记忆增强规划**。该策略通过引入一个附加的记忆组件来提升规划能力，该组件中存储了各种宝贵信息，包括基本常识、历史经验、领域专业知识等。在进行规划时，这些信息会被调取出来，充当辅助提示，以增强规划的效果。分为RAG记忆（即接下来要介绍的Memory）和嵌入式记忆（将RAG知识通过微调嵌入到模型参数里）

Table 1: A taxonomy for existing LLM-Agent planning works.

Method	Idea	LLM's task	Formulation	Representative works
Task Decomposition	Divide and Conquer	Task decomposition Subtask planning	$[g_i] = \text{decompose}(E, g; \Theta, \mathcal{P})$; $p^i = \text{sub-plan}(E, g_i; \Theta, \mathcal{P})$	CoT [2022], ReAct [2022], HuggingGPT [2023]
Multi-plan Selection	Generate multiple plans and select the optimal	Plans generation Plans evaluation	$P = \text{plan}(E, g; \Theta, \mathcal{P})$; $p^* = \text{select}(E, g, P; \Theta, \mathcal{F})$	ToT [2023], GoT [2023], CoT-SC [2022b]
External Planner-aided	Formalize tasks and utilize external planner	Task formalization	$h = \text{formalize}(E, g; \Theta, \mathcal{P})$; $p = \text{plan}(E, g, h; \Phi)$	LLM+P [2023a], LLM+PDDL [2023]
Reflection & Refinement	Reflect on experiences and refine plans	Plan generation Reflection Refinement	$p_0 = \text{plan}(E, g; \Theta, \mathcal{P})$; $r_i = \text{reflect}(E, g, p_i; \Theta, \mathcal{P})$; $p_{i+1} = \text{refine}(E, g, p_i, r_i; \Theta, \mathcal{P})$	Reflexion [2023], CRITIC [2023], Self-Refine [2023]
Memory-aided Planning	Leverage memory to aid planning	Plan generation Memory extraction	$m = \text{retrieve}(E, g; \mathcal{M})$; $p = \text{plan}(E, g, m; \Theta, \mathcal{P})$	REMEMBER [2023a], MemoryBank [2023]

Memory

记忆模块是智能体存储内部日志的关键组成部分，负责存储过去的思考、行动、观察以及与用户的互动。

- **短期记忆**关注于当前情境的上下文信息，是短暂且有限的，通常通过上下文窗口限制的学习实现。
- **长期记忆**储存智能体的历史行为和思考，通过外部向量存储实现，以便快速检索重要信息。
- **混合记忆** -通过整合短期和长期记忆，不仅优化了智能体对当前情境的理解，还加强了对过去经验的利用，从而提高了其长期推理和经验积累的能力。

记忆存储到外部存储器中，最常见的做法是将记忆的embedding存储到支持快速的最大内积搜索（MIPS）的向量存储数据库中。不只是文本，图像、音视频等非结构化数据也可以存储为结构化向量，降低了存储和计算的成本，同时加速了检索效率。

基本概念

- 任务：agent要实现的目标，例如订一个机票，下面用 $\mathcal{T}$ 表示一个问题。
- 环境：agent为了完成任务需要与环境交互，环境包含了可能改变agent决策的上下文信息。
- trail：agent采取行动，并从环境获得行动的反馈，基于此反馈再采取行动，循环持续直到任务完成，这一过程被称为trail。一个长度为T的trail可以表示为 $\xi_T = a_1, o_1, ..., .a_T, o_T$ ，其中 a和o 分别是行动和环境。每一轮agent与环境的交互被称为一个step。**每个任务可能有多个trail，即为了完成一个任务可能做很多尝试。**
- memory：狭义上的memory指同一个trail中的历史信息。给定一系列任务 $\{\mathcal{T}_1, ..., \mathcal{T}_k\}$ ，对于任务 $\mathcal{T}_k$ 在第t步时，广义的memory来自3个方面：

📌
- 1）同一个trail之前的历史信息，记为：
- 2）任务k之前任务的trail，以及任务k之前的trail（记为k'）：
- 3）外部知识，比如通过RAG查询到的知识，表示为：

$$\xi_t^k = \{a_1^k, o_1^k, ..., a_{t-1}^k, o_{t-1}^k\}$$

$$\Xi^k = \{\xi^1, \xi^2, ..., \xi^{k-1}, \xi^{k'}\}$$

$$D_t^k$$

Memory-based agent

agent在与环境交互的过程可以分为3个阶段。

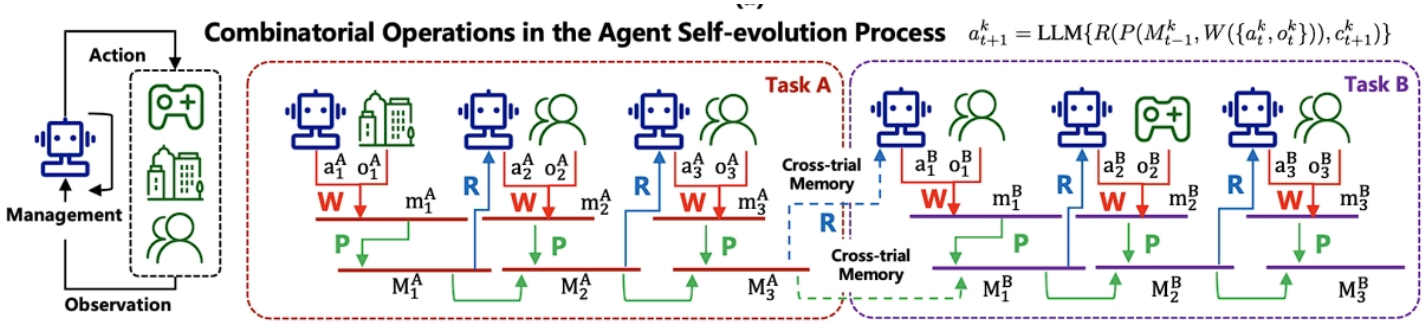
- 1）智能体从环境中感知信息，并将其存储到记忆中。
- 2）智能体对存储的信息进行处理，使其更加可用；
- 3）智能体根据处理后的记忆信息采取下一步行动。

对应了三个agent memory的三个操作：

- **写记忆：**此操作旨在将从环境的原始观察结果投射到实际存储的记忆内容中，这些内容更具信息量和简洁。这一操作可以表示为 $m_t^k = W(\{a_t^k, o_t^k\})$，W表示映射函数，$m_t^k$ 是最终写入memory的内容，可以以自然语言的形式或者参数化的形式。
- **记忆管理：**使得记忆信息更加高效，例如总结高级的概括性概念以使得agent更具有泛化性，合并相似信息以降低冗余性，忘记不重要信息避免造成负面影响。这一操作表示为 $M_t^k = P(M_{t-1}^k, m_t^k)$，$m_t^k$ 是第t轮的记忆，M_{t-1}^k 表示之前处理过的记忆，P是迭代处理存储的记忆信息的函数。对于广义的记忆，这一操作会跨trail跨任务执行，并且会随着外部知识的变化而执行。
- **读记忆：**从memory获取信息以采取下一次行动，$\hat{M}_t^k = R(M_t^k, c_{t+1}^k)$，$c_{t+1}^k$ 表示下一个行动的上下文，R是计算相似度的函数，$\hat{M}_t^k$ 表示计算得到的最相似的记忆内容，会被加入下一轮agent的prompt中。

基于以上操作，就可以得到agent做决策的统一表示，下图展示了agent完成两个任务的工作流程。

$$a_{t+1}^k = \text{LLM}\{R(P(M_{t-1}^k, W(\{a_t^k, o_t^k\})), c_{t+1}^k)\}$$



为什么要使用Memory-based agent

- 从认知心理学角度：对于人类的认知来说，记忆重要的模块，agent想要替代人类完成一些任务，就要表现的像人类，为agent设置代理模块
- 从自我进化角度：在完成任务的过程中，agent也需要在与环境交互时自我进化。记忆能帮助agent积累经验、探索更多的环境、抽象出概括性信息以增强泛化性。
- 从agent应用角度：在很多应用中agent不可取代的，例如chatgpt、虚拟角色。

参考：**A Survey on the Memory Mechanism of Large Language Model based Agents**

先插播一下langchain中使用memory的例子，后续会更新这篇文章的后半部分：创建一个简单的自定义代理，该代理可以访问搜索工具并使用 `ConversationBufferMemory` 类。首先定义搜索工具

```
from langchain.agents import ZeroShotAgent, Tool, AgentExecutor
from langchain.memory import ConversationBufferMemory
from langchain import OpenAI, LLMChain
from langchain_core.tools import Tool
from langchain_community.utilities import SerpAPIWrapper
from langchain_openai import ChatOpenAI
import os
os.environ["SERPAPI_API_KEY"] = (
    "Your serpapi key"
)
search = SerpAPIWrapper()
from langchain.agents import load_tools
llm = ChatOpenAI(model=os.environ["LLM_MODELEND"], temperature=0)
tools = load_tools(["serpapi"], llm=llm)
```

接下来定义一个prompt，prompt中的的chat_history需要与ConversationBufferMemory的key对应，存储对话记录。

```
prefix = """Have a conversation with a human, answering the following
questions as best you can. You have access to the following tools:"""
suffix = """Begin!"

{chat_history}
Question: {input}
{agent_scratchpad}"""

prompt = ZeroShotAgent.create_prompt(
    tools,
    prefix=prefix,
    suffix=suffix,
    input_variables=["input", "chat_history", "agent_scratchpad"],
)
memory = ConversationBufferMemory(memory_key="chat_history")
```

接下来创建一个LLMChain，然后将momory加入到agent中。

```
llm_chain = LLMChain(llm = llm, prompt=prompt)
```

```
agent = ZeroShotAgent(llm_chain=llm_chain, tools=tools, verbose=True)
agent_chain = AgentExecutor.from_agent_and_tools(
    agent=agent, tools=tools, verbose=True, memory=memory
)

agent_chain.run(input="How many people live in canada?")
agent_chain.run(input="what is their national anthem called?")
```

langchain中为我们实现了多种记忆，常见的有：

ConversationBufferMemory：所有聊天记录都被存入chat_history中，导致下一轮的prompt很长

ConversationBufferWindowMemory：只保留最近几次人类与AI的互动，只适应短对话。

ConversationSummaryMemory：在回答新问题的时候，对之前的问题进行了总结性的重述，再传递给chat_history 参数，这种基于总结的方法能避免过度使用token，适合长对话。总结由LLM完成，虽然最初使用的 Token 数量较多，但随着对话的进展，汇总方法的增长速度会减慢；并且，总结的过程中并没有区分近期的对话和长期的对话（通常情况下近期的对话更重要）。

ConversationSummaryBufferMemory：总结较早的对话，保留近期的对话原始内容。