agent

2023년 10월 30일 월요일 오전 7:18

Agents | □□ Langchain

agent type,how to, tools, toolkits, callbacks

agent : 행동할 action의 순서를 llm한테 알려주는것이 핵심이다 : tool,meomorry 연결하고 llm을 reasoning engine역할로 사용하는게 일반적이다

chains : a sequence of actions

agent : resoning engine to determine which actions to take and in which order : 어떤 순서로 어떤 액션할지 resoning engine한테 넘긴다

서로 다른 agent는 서로 다른 prompt를 가지고 있다 (for resoning) 즉 다른 encoding input, 다른 outputparser를 각각 가지고 있다는 의미이다

보통은 React기반으로 agent를 작동한다
간단한 작동 방식

- agent한테 task주기
- Thought : the agent thinks about what to do
- Action/Action Input : The agent decides what action to take (어떤 tool사용할지 정하는거)
- observation : the output of the tool

tool만들때 LLMChain 통해 llm과 prompt정의 해주면 custom tool만들때 run할수있다

input

1. list of available tools
    a. Giving the agent access to the right tools
    b. describing the tools in a way that is most helpful to the agent
    c. 보통 @tool decorator를 사용해서 tool정하는것 같은데 이건 확인을 해봐야 할것 같다
2. user input
3. any previously executed steps(intermediate_steps)

return
AgentAction or AgentFinish

BaseTool을 이용해서 tool만들때 _run함수 만들어야 tool선언할때 사용 가능

set of tools => toolkits을 지원하기도 한다

important schema

1. AgentAction
    a. agent가 해야할 action의 dataclass -> tool property and tool input prperty
2. AgentFinish
    a. agent가 return해야할것을 알려주는 dataclass -> return values parameter, output
3. itermediate_steps
    a. 이전 agent결과이다

기본적으로 agent에서 적용할 prompt(react,zeroshot,등등) 여기서 pull해서 사용한다
LangSmith (langchain.com)

AgentExecutor

runtime for an agnet
Agent Executors take an agent and tools and use the agent to decide which tools to call and in what order.

이걸 사용하면 여러가지 상황을 handing할수있다

1. agent가 존재하지 않는 tool사용한다거나
2. tool errors있다거나
3. agent의 output이 parser가 안되는 문제 있다거나
4. logging and observability at all levels 등등

AgetnAction, observation 반복하다가 AgentFinish나오면 user한테 return 한다

other types of agent runtimes

- plan and execute agent
- baby AGI
- Auto GPT

Agent Types

Agent Types | ☐☐ Langchain

conversational

Conversational | ☐☐ Langchain

보통 agent는 어떤 tool을 사용할지에 optimized되어있다면 conversation은 Agent optimized for conversation 이다

predefined agent type을 이용해서 agent executor 초기화 할수도 있다

ReAct

ReAct: Synergizing Reasoning and Acting in Language Models (react-lm.github.io)

ReAct | ☐☐ Langchain

from langchain import hub

hub.pull("hwchase17/react") -> react,zeroshot등 prompt가져 올수있다

agent보면 inputm agent_scratchpad를 넣어야 한다

ZeroShotReactAgent

initialize_agent(tools,llm,agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION)

React document store

ReAct document store | ☐☐ Langchain
document 기반으로 react적용할수있다는 예시이다

self ask with search

Agent that asks intermediate questions, but answers them with a search tool

structured tool chat

Structured tool chat | ☐☐ Langchain
multi input tools에 사용한다

XML Agent

XML Agent | ☐☐ Langchain

XML이란 무엇인가요? - Extensible Markup Language(XML) 설명 - AWS (amazon.com)
xml(extensible markup language) : 공유 가능한 방식으로 데이터 정의하고 저장할수있다. 웹사이트, 데베,밑 컴퓨터 시스템간의 정보교환 지원한다.

데이터를 정의하는 규칙을 제공하는 마크업 언어 그래서 구조적 데이터 관리를 위해 구현 가능하다

XML에서 태그라고 하는 마크업 기호를 사용하여 데이터를 정의합니다. 예를 들어, 서점에 대한 데이터를 나타내기 위해 <book>, <title> 및 <author>와 같은 태그를 만들 수 있습니다. 책 한 권의 XML 문서에는 다음과 같은 내용이 포함됩니다.

```
<book>
<title> Learning Amazon Web Services </title>
<author> Mark Wilkins </author>
</book>
```

태그는 정교한 데이터 코딩을 통해 여러 시스템에서 정보 흐름을 통합합니다.

XMLAgent import해서 default_prompt와 degault_output_parser를 사용해서 결과 만들수있다

How to

add memory to agent

Add Memory to OpenAI Functions Agent | ☐☐ Langchain
serarch tool, math tool과 converstation memorry 사용해서 agent 구현

running agent as an iterator

Running Agent as an Iterator | ☐☐ Langchain

problem

- retrieve three prime numbers from a tool
- multiply these together

AgentExecutorIterator :

하고 있는 예제를 보면 agent에서 나온 intermiediate step결과를 가지고 사용자와 상호작용한것도 확인할수있다

intermiediate step의 type : intermediate_steps: List[Tuple[AgentAction, str]]

returning structured output

string으로 결과를 받는대신 원하는 structed형태로 ouptut출력할수있다

```
from langchain.agents.agent_toolkits.conversational_retrieval.tool import (
    create_retriever_tool,
)

retriever_tool = create_retriever_tool(
    retriever,
    "state-of-union-retriever",
    "Query a retriever to get information about state of the union address",
)
```

화면 캡처: 2023-11-04 오전 8:19

다음과 같이 import한 create_retriever_tool을 사용해서 tool만든다

AgentAction을 사용하는것 보다 AgentActionMessageLog를 사용하면 log정보 같이 볼수있다

combine agents and vector stores

agent와 vector store combine : retrival QA를 tool로 만들어서 agent로 사용한다

예제에서는 doc기반 retrival QA와 web기반 retrival QA를 tool로 만들어서 agent로 사용하고 있다
위에 예제에서는 querying이후 추가 작업을 한게 나오는데(thoght,observation) tool만들때 return_direct =True를 해서 result를 바로 받게 할수있다
물론 multi hop vector store reasoning가능하기 때문에 하나의 질문 말고 여러 개의 질문을 동시에 해도 괜찮다

async API

Async API | ☐☐ Langchain

asyncio library를 통해 지원한다
[미완]

create chatgpt clone

Create ChatGPT clone | ☐☐ Langchain

[미완]

custom functinos with agent

Custom functions with OpenAI Functions Agent | ☐☐ Langchain
[미완]

cusom agent

custom agent만들때 고려해야할것은 2가지이다

- tools : agent가 사용할 tools
- agent class itself : 어떤 action을 할지정해야한다

BaseSingleActionAgent를 상속 받아서 class만든다
plan함수를 만들었고 사용하는데는 없는데 왜 작동하는지는 잘 모르겠다

custom agent with tool retrieval

Custom agent with tool retrieval | ☐☐ Langchain

진짜 tool하나와 fake tool99개를 합쳐서 100개의 tool을 넣은 agent구성을 하는데 원래 같으면 agent에서 어떤 tool사용하고 action할지를 정해야 하는데
그렇게 기존처럼 하는대신 tools를 retirval vector store에 넣어서 query와 비슷할것 같은 tool을 찾는 get_tools함수를 만들어서 사용한다는 개념 같다

custom llm agent
Custom LLM agent | □□ Langchain

CustomPromptTemplate -> def format정의해서 intermediate steps을 얻고 format한다
CusomOutputParser -> def parse를 정의해서 다룬다

custom llm agent(with chatmodel)
Custom LLM Agent (with a ChatModel) | □□ Langchain
위와 비슷하다

custom MRKL agent
Custom MRKL agent | □□ Langchain

MRKL(modular reasoning,knowledge and language)
combines llm,extract knowledge sources,discrete reasoning

zeroshotagent.create_prompt 사용한다 prefix(이전내용)와 surffix(이후내용)

custom multi action agent
Custom multi-action agent | □□ Langchain

custom agent를 만들때 def plan에 AgentAction을 여러 개 사용해서 여러 개의 agent가 발생하도록 한 예제였다

handle parsing errors
Handle parsing errors | □□ Langchain

initialize_agent선언할때 arg로 handle_parsing_errors=True로 설정하면 default error handling할수가 있다
[미완 : custom error handling을 따로 가서 확인해라]

access intermediate steps
Access intermediate steps | □□ Langchain

intermediate step을 return 하면서 agent가 무엇을 하는지 확인 할 수가있다

initialize_agent할때 arg로 return_intermediate_steps=True로 설정하면 된다
```
print(response["intermediate_steps"])
```

출처: <https://python.langchain.com/docs/modules/agents/how_to/intermediate_steps>

```
from langchain.load.dump import dumps

print(dumps(response["intermediate_steps"], pretty=True))
```

cap the max number of iterations
Cap the max number of iterations | □□ Langchain

initialize_agent max_iterations을 설정하는 방법

timeouts for agents
Timeouts for agents | □□ Langchain
long running agent runs를 방지하는 timeout기능

replicating MRKL
Replicating MRKL | □□ Langchain

agent를 사용해서 MRKL 모사하는 방법에 대해서 다룬다

shared memory across agents and tools
Shared memory across agents and tools | □□ Langchain

conversation memory에 접근해서 사용할수있게 하는 예제이다 -> ReadOnlySharedMemory

streaming final agent output

Streaming final agent output | ☐☐ Langchain

final output of an agent -> streamed

use toolkit with functions
Use ToolKits with OpenAI Functions | ☐☐ Langchain

지원하는 toolkit을 사용하는 방법에 대한 설명이다

Tools

defining custom tools
Defining Custom Tools | ☐☐ Langchain

- name
- description
- return_direct : default False
- args_schema(pydantic basemodel)

custom tool만드는 방법

pydantic BaseModel 사용
pydantic의 BaseModel을 사용해서 class만들어서 argument schema를 작성한다
작성한 class를 다시 basemodel사용해서 def _run함수 정의해서 다룰수도 있다 진짜 custom으로 하려면 함수 작동이 있어야 한다 예제에서는 그냥 있는 tool에 run한거 적용함

@tool 데코레이터 사용
overriding해서 first argument로 넘기는것이다

custom structured tools

```
import requests
from langchain.tools import StructuredTool


def post_message(url: str, body: dict, parameters: Optional[dict] = None) -> str:
    """Sends a POST request to the given url with the given body and parameters."""
    result = requests.post(url, json=body, params=parameters)
    return f"Status: {result.status_code} - {result.text}"


tool = StructuredTool.from_function(post_message)
```

human in the loop tool validation
Human-in-the-loop Tool Validation | ☐☐ Langchain

HumanApprovalCallbackhandler : 말그대로 사람이 직접 승인 해주는 callback이다

multiple agent에서 특정 tool에 대한 입력만 human approval하고 싶다면 다시 사이트 참고 해라

multi input tools
Multi-Input Tools | ☐☐ Langchain

tool input schema
Tool Input Schema | ☐☐ Langchain

tool as functions

toolkit
Agent Toolkits (langchain.dev)

callbacks

logging,monitoring,streaming, and other task

callbackHandlers

Function calling

how to connect llm to external tools : 외부 tool과 llm연결 방법

open ai llm은 functions를 통해 arbitary entities를 추출할수가 있다

```
function_tools = [{"type": "function",
        "function":{
            'name': 'information_extraction',
            'description': 'Extracts the relevant information from the passage.',
            'parameters': {
                'type': 'object',
                'properties': {
                    'info': {
                        'type': 'array',
                        'items': {
                            'type': 'object',
                            'properties': {
                                'name': {'title': 'name', 'type': 'string'},
                                'height': {'title': 'height', 'type': 'integer'},
                                'hair_color': {'title': 'hair_color', 'type': 'string'}
                            },
                            'required': ['name', 'height']}},
                },
                'required': ['info'] }}}]
```

화면 캡처: 2023-12-15 오후 12:02

llm = ChatOpenAI(temperature=0,model=model).bind(tools=function_tools) -> 모든 llm이 되는건 아니다
즉 pydantic output parser만 지원 하는게 아니다

---산학 과정적용 생각 -----

get_current_weather(location) -> wether api를 사용해서 location의 날씨 정보를 가져오는 외부 api이다

```
function_description = [
    {
        "name": "get_current_weather",
        "description": "Get the current weather in a given location",
        "parameters": {
            "type": "object",
            "properties": {
                "location": {
                    "type": "string",
                    "description": "The city and state, e.g. San Francisco, CA"
                },
                "unit": {
                    "type": "string",
                    "enum": ["celsius", "fahrenheit"],
                    "description": "The temperature unit to use. Infer this from the users location.",
                }
            },
            "required": ["location", "unit"]
        }
    }
]
```

출처: <http://127.0.0.1:63342/jupyter/index.html>

다음 처럼 args type과 description설정하여 만든 함수를 적용할수있다

```
essages = [
    {"role": "user", "content": "What is the weather like in Seoul?"}
]
response = openai.chat.completions.create(
    model = "gpt-4-1106-preview",
    messages = messages,
    functions = function_description,
)
```

출처: <http://127.0.0.1:63342/jupyter/index.html>

다음 처럼 openai.chat.completions.create를 통해 llm설정과 functions를 설정한다
functions는 앞에서 만든 get_current_weather함수이다 이를 활용해서 적용할수있다

```
response.choices[0].message
ChatCompletionMessage(content=None, role='assistant', function_call=FunctionCall(arguments='{"location":"Seoul","unit":"celsius"}', name='get_current_weather'), tool_calls=None)
```

출처: <http://127.0.0.1:63342/jupyter/index.html>

```
print(weather)
weather_str = str(weather)
{'temperature': 1.0, 'humidity': 34, 'weather_description': 'Sunny'}
```

```
[9]
messages = [
    {"role": "user", "content": "What is the weather like in Seoul?"}
]
messages.append({'role':'function', "name": func_name, 'content': weather_str})
messages
[{'role': 'user', 'content': 'What is the weather like in Seoul?'},
 {'role': 'function',
  'name': 'get_current_weather',
  'content': "{'temperature': 1.0, 'humidity': 34, 'weather_description': 'Sunny'}"}]
```

출처: <http://127.0.0.1:63342/jupyter/index.html>

open ai model사용하면 위에 기능 지원하지만 llama2같이 다른 llm 사용시 지원하지 않는다
together.complet.create는 있는데 functions는 지원하지 않아서 사용 못함

open ai 사용시 papago api를 functions로 묶어서 사용할수있을것 같다
하지만 data번역시 source 와 target언어를 설정해야하는데 이전에 f string으로 처리하도록 하였을때 http 400error가 발생하여서
결국 한->영 , 영 -> 한을 각각 함수로 만들었었다. open ai 모델 functions를 통해 위의 문제를 해결할수있을지는 직접 해보아야 할것 같다
모든것을 떠나서 외부 api를 functions로 묶어서 처리 할 수 있는이건 큰 장점인것 같다
그리고 json형식으로 parsing도 쉽게 이루어지는것 같다

애초에 라마2이용했을때 번역 한것인데 open ai 모델이라면 한국어도 어느정도 하니까
번역하는 prompt만 적용해도 파파고 번역은 쉽게 이룰수있었을것 같다

Parsing

```
class Person(BaseModel):
person_name: str
person_height: int
person_hair_color: str
dog_breed: Optional[str]
dog_name: Optional[str]

class People(BaseModel):
"""Identifying information about all people in a text."""

people: Sequence[Person]

# Set up a parser + inject instructions into the prompt template.
parser = PydanticOutputParser(pydantic_object=People)
```

화면 캡처: 2023-12-15 오후 12:04

create_extraction_chain

```
# Schema
schema = {
    "properties": {
        "name": {"type": "string"},
        "height": {"type": "integer"},
        "hair_color": {"type": "string"},
    },
    "required": ["name", "height"],
}

# Input
inp = """Alex is 5 feet tall. Claudia is 1 feet taller Alex and jumps higher than him. Claudia is a brunette and
Alex is blonde."""

# Run chain
llm = ChatOpenAI(temperature=0, model="gpt-3.5-turbo")
chain = create_extraction_chain(schema, llm)
chain.run(inp)
```

화면 캡처: 2023-12-15 오후 12:04

산학 agent 에 대한 구상 고민 시작 [autonomous agents,plan executer,Reflecting]

Hallucination현상을 완화 하려는 시도 영상

(54) Chain-of-Verification: How to fight AI Hallucination - YouTube

fight halucinate method

- joint method
- 2 step method : planning prompt + execution prompt
  - planning prompt: Generate verification questions to fact-check the information about the {query}
  - execution prompt : Answer the following {verificaiton questions(planning에서 나온결과)}
- factor + revise method
  - 2 step method에서 나온 결과 물에서 한번 더 들어간다
  - revise prompt : cross check the {answers(execution 결과)} for inconsistencies with the {planning 프롬프트결과}

Enhance the prompting techniques

- hierarchical prompting : 계층적으로 프롬프트를 강화 하는 방식
- conditional prompting : if else처럼 조건부 논리를 프롬프트에 적용하는 방식
- confidence prompting : LLM한테 이전에 내놓은 답변이 1-10점까지 중에 얼마나 자신있는지 평가하게 한다

COT->GOT

(55) Graph-of-Thoughts (GoT) for AI reasoning Agents - YouTube
Lil'Log (lilianweng.github.io)

node : llm thoughts
edge : connectome

multi ai agents resoning llm

Multi AI-Agents Reasoning LLM - CODE Examples (Python)

[미완 : 코드 참고할만한것 같진않아서이다]

CAMEL

Autonomous Agents & Agent Simulations (langchain.dev)

main: 두개의 agent가 interaction하는것이다 핵심은 동일선상에서 interaction하는게 기존과 다르다
largely reflecting the simulation environment

plan and execute agents

Plan-and-Execute Agents (langchain.dev)

기존 react방식의 agent(action agent)는 agent에 의존적이고 observation은 더 복잡한 상황에서는 애매한 문제있다

observation이 더 복잡해진 만큼 이전 스텝의 resoning과 remember더 해야 한다

implementation

- Plan steps to take
- For step in steps: determine the proper tools or other best course of action to accomplish that step

화면 캡처: 2023-11-05 오후 3:01

planner 와 executor에 의존한다

보통 planner는 llm을 의미한다 llm은 plan하기 위한 resoning 능력을 통합할것이다 그리고 애매하거나 edge case를 다룰수있다

executor : action agent..??

Reflection==Reflexion

Reflexion is all you need?. Things are moving fast in LLM and... | by Jens Bontinck | ML6team
2303.11366.pdf (arxiv.org)

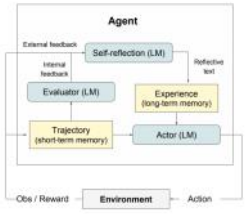
Hallucination : 말도 안되는 내용을 생성하는 것을 의미한다 : predict next world를 하기 때문에 이런 실수가 나오는것이다

Reflection의 목적은 이러한 hallucination을 완화 하는것이다 : 언어의 반영을 통해 강화 한다

"LLM-in-the-loop" instead of "human-in-the-loop" approach.

예를 들어서 어떤 작업 완성한다 할때 결과물을 다시 읽어서 적당한지 판단하는 작업을 하잖아 아니라면 다시 수정하는 작업을 하는것 처럼 reflection도 이와 같은 메커니즘을 반영한것이다

Ma(actor) : 구체적인 prompt를 장착한 LLM
Me(evaluator) : actor에서 나온 결과를 crucial 하는 역할이다 semantic space에 대한 평가는 어렵다 그래서 다양한 evaluator model사용한다는데 자세한건 모르겠다
self reflection : complex task를 배울때 유용하다. LLM으로 instance한다



Algorithm 1 Reinforcement via self-reflection

Initialize Actor, Evaluator, Self-Reflection: M_a, M_e, M_{sr}
Initialize policy $\pi_\theta(a_i|s_i), \theta = \{M_a, mem\}$
Generate initial trajectory using π_θ
Evaluate τ_0 using M_e
Generate initial self-reflection sr_0 using M_{sr}
Set $mem \leftarrow [sr_0]$
Set $t = 0$
while M_e not pass or $t <$ max trials **do**
  Generate $\tau_t = [a_0, o_0, \dots a_i, o_i]$ using π_θ
  Evaluate τ_t using M_e
  Generate self-reflection sr_t using M_{sr}
  Append sr_t to mem
  Increment t
end while
return

Figure 2: (a) Diagram of Reflexion. (b) Reflexion reinforcement algorithm

화면 캡처: 2023-11-05 오후 5:59

self -RAG : 다음 영상은 self RAG영상인데 여기서 self reflection개념과 semantic evalutte에 관한 내용이 있다 여기서 관련 개념을 익힌다
Self-RAG: Learning to Retrieve, Generate and Critique through Self-Reflection (selfrag.github.io)
hugging face에 self rag llama2-7b모델 사용 가능하다

RCI(Recursive Criticism and Improvement) chain

위에서 보았던 reflexion과 상당히 비슷한 부분이 많이 있다
Building a RCI Chain for Agents with LangChain Expression Language

결과 -> 비판 -> 원래 질문,결과,비평 모두 하나의 프롬프트로 만들어서 출력하는 과정이다 (Recursive라서 여러 번 과정을 반복 할수있다)

intial chain->critique chain-> improvement chain