Async callbacks

If you are planning to use the async API, it is recommended to use `AsyncCallbackHandler` to avoid blocking the runloop.

Advanced if you use a sync `CallbackHandler` while using an async method to run your llm/chain/tool/agent, it will still work. However, under the hood, it will be called with `run_in_executor` which can cause issues if your `CallbackHandler` is not thread-safe.

```
import asyncio
from typing import Any, Dict, List

from langchain.chat_models import ChatOpenAI
from langchain.schema import LLMResult, HumanMessage
from langchain.callbacks.base import AsyncCallbackHandler,
BaseCallbackHandler


class MyCustomSyncHandler(BaseCallbackHandler):
    def on_llm_new_token(self, token: str, **kwargs) -> None:
        print(f"Sync handler being called in a `thread_pool_executor`:
token: {token}")


class MyCustomAsyncHandler(AsyncCallbackHandler):
    """Async callback handler that can be used to handle callbacks from
langchain."""

    async def on_llm_start(
        self, serialized: Dict[str, Any], prompts: List[str], **kwargs:
Any
    ) -> None:
        """Run when chain starts running."""
        print("zzzz....")
        await asyncio.sleep(0.3)
        class_name = serialized["name"]
        print("Hi! I just woke up. Your llm is starting")

    async def on_llm_end(self, response: LLMResult, **kwargs: Any) ->
None:
        """Run when chain ends running."""
        print("zzzz....")
```

```
        await asyncio.sleep(0.3)
        print("Hi! I just woke up. Your llm is ending")


# To enable streaming, we pass in `streaming=True` to the ChatModel
constructor
# Additionally, we pass in a list with our custom handler
chat = ChatOpenAI(
    max_tokens=25,
    streaming=True,
    callbacks=[MyCustomSyncHandler(), MyCustomAsyncHandler()],
)

await chat.agenerate([[HumanMessage(content="Tell me a joke")]])
```

API Reference:

- ChatOpenAI from `langchain.chat_models`
- LLMResult from `langchain.schema`
- HumanMessage from `langchain.schema`
- AsyncCallbackHandler from `langchain.callbacks.base`
- BaseCallbackHandler from `langchain.callbacks.base`

```
    zzzz....
    Hi! I just woke up. Your llm is starting
    Sync handler being called in a `thread_pool_executor`: token:
    Sync handler being called in a `thread_pool_executor`: token: Why
    Sync handler being called in a `thread_pool_executor`: token:  don
    Sync handler being called in a `thread_pool_executor`: token: 't
    Sync handler being called in a `thread_pool_executor`: token:
scientists
    Sync handler being called in a `thread_pool_executor`: token:
trust
    Sync handler being called in a `thread_pool_executor`: token:
atoms
    Sync handler being called in a `thread_pool_executor`: token: ?
    Sync handler being called in a `thread_pool_executor`: token:


    Sync handler being called in a `thread_pool_executor`: token:
Because
    Sync handler being called in a `thread_pool_executor`: token:  they
    Sync handler being called in a `thread_pool_executor`: token:  make
    Sync handler being called in a `thread_pool_executor`: token:  up
```

```
    Sync handler being called in a `thread_pool_executor`: token:
everything
    Sync handler being called in a `thread_pool_executor`: token: .
    Sync handler being called in a `thread_pool_executor`: token:
    zzzz....
    Hi! I just woke up. Your llm is ending



    LLMResult(generations=[[ChatGeneration(text="Why don't scientists
trust atoms? \n\nBecause they make up everything.",
generation_info=None, message=AIMessage(content="Why don't scientists
trust atoms? \n\nBecause they make up everything.", additional_kwargs=
{}, example=False))]], llm_output={'token_usage': {}, 'model_name':
'gpt-3.5-turbo'})
```