OpenAI API

OA1- Installing the OpenAI Module

Lesson Objectives:

- Understand how to use `pip` to install the OpenAI package in order to connect to the LLM

[VIDEO] https://youtu.be/DiJ1XB79u4Y

Lesson Notes

To use the OpenAI API:

In the students Colab Notebook, you will see:

```
!pip install openai
```

This means that we will be using `pip`- which stands for Python Installation Package in order to install the OpenAI package and import their funcitonalities with this statement below:

```
from openai import OpenAI
from google.colab import userdata
```

With this statement, we will be importing the OpenAI functionalities stored within the openai module which is essential for us to communicate with the model

OA2 - Generating an OpenAI API Key

- Understand the importance of using and securing an API Key to connect to the OpenAI LLM's

[VIDEO]: https://youtu.be/AJohBn_louY

Lesson Notes

Head over to `https://platform.openai.com/api-keys`

- Click on the "Create a new Secret Key"
- Copy the API Key to your clipboard by pressing the green button

OA3 - Using the OpenAI API Key

Lesson Objective:

- Understand how to use Colab Secrets to store the OpenAI API Key for safe usage and sharing of Python Notebooks

[Video] https://youtu.be/UELnZu6Al1c

Lesson Notes

We will be using Colab Secrets in order to store our API Keys for us to not hardcode the key into the Colab File to prevent our API Key from being leaked.

![image-20240424101733392](/Users/a1234/Library/Application Support/typora-user-images/image-20240424101733392.png)

Click on the key symbol (shown on the left of the picture above) and click on `+ Add new Secret`

For the name section fill in: `OPENAI_API_KEY` and the value should be key you have copied from the previous lecture (e.g. `sk-AAAAAAAAAAAAA`)

OA-4 - Prompting with the OpenAI API

Lesson Objective:

- Understand the basic syntax of using the OpenAI API
- Understand how to perform basic user prompting with the OpenAI API

[VIDEO]: https://youtu.be/IzkhoWIvMOk

We will first need to initialise a "connection" to the OpenAI LLM with the variable `client` holding the connection:

```
client = OpenAI(api_key = userdata.get("OPENAI_API_KEY"))
```

We specifiy our own OpenAI API Key by passing in the `api_key = userdata.get("OPENAI_API_KEY")` to retrieve our API Key which we have generated in the previous lectures

Now that we have a connection with the OpenAI API Key, we are now able to get a response from OpenAI with a `response` variable holding the output of the response:

```
response = client.chat.completions.create(
        model = "gpt-3.5-turbo",
    messages = [{'role': 'user', 'content': 'How should we use AI responsibly'}]
)
```

We must pass in two mandatory arguments: `model` and `messages` which stores:

`model`: `"gpt-3.5-turbo"` / `"gpt-4-turbo"`

`messages`: a list of dictionaries containing the two keys, `role` (in this case user as we are asking questions) and `content` (which is our question in this case) (e.g. `[{'role': 'user', 'content': 'How should we use AI responsibly'}]`)

In order for us to access the response we need to dissect the `ChatCompletions` object from response in this manner:

```
print(response.choices[0].messages.content)
```

OA-5: OpenAI Roles

Lesson Objectives:

- Understand how to use the 3 different roles, `system` , `user` and `assistant` when prompting with the OpenAI API and its significance

[VIDEO] https://youtu.be/zL0sboGAKIE

In the previous lecture, we have taken a look at how we can do basic prompting with the OpenAI API. However, there is more flexibility with using the API than the actual chatbot as we are able to control the direction of the conversation with the `systems` role, ask questions with the `user` role and put model response as the `assistant role`

Lesson Notes

System Role:

- Control the model interactions more definitively than the user role
- GIve high level directions for the conversation
- Used to set the type of response for the conversation (i.e. we will specify how the model should reply here with tone & persona)

User Role:

- Similar to the Chat Interface we use in the ChatGPT website
- Prompt Engineer your prompts here

Assistant Role:

- Serves as memory to keep track of the conversation
- Create replies that are coherent and contextually relevant to both user and system roles
- Direction of response is controlled by the System Role

With `System and User Role`:

We expand the list to add a dictionary containing the role of system and give it persona, then we can continue the conversation

```
response = client.chat.completions.create(
    model = 'gpt-3.5-turbo',
    messages = [{'role': 'system', 'content': 'You are a Singaporean'},
                {'role': 'user', 'content': 'When is your nation"s birthday?'}
```

```
                    ]
)
print(response.choices[0].message.content)
```

With `System, User and Assistant Role`:

After receving the model response, we can add the response back to the list of the messages with a dictionary holding the role of assistant and the key idea of the model response

```
response = client.chat.completions.create(
    model = 'gpt-3.5-turbo',
    messages = [{'role': 'system', 'content': 'You are a Singaporean'},
                {'role': 'user', 'content': 'When is your nation"s birthday?'},
                {'role': 'assistant', 'content': 'Singapore\'s National Day is 
celebrated on August 9th.'},
                {'role': 'user', 'content': 'How many days is it away from 
Teachers Day?'}
                ]

)
print(response.choices[0].message.content)
```

OA6 - Hyperparamter Tuning

Lesson Notes

- Understand the significance of using the following hyperparamters: `temperature`, `frequency_penalty` and `presence_penalty` when using the OpenAI API and how it affects the model output

[VIDEO] https://youtu.be/aupOgCvfYXE

We can also control the creativity of the model output by passing in hyperparameters when using the OpenAI API.

These are the hyperparameters that we will be covering in this lecture in order to change the model responses:

`temperature`: Controls the randomness of the AI response. Default value is 1 while it ranges from 0.0 to 2.0

`frequency_penalty`: Penalise the response based on their existing frequency (Deafults to 0, ranges from -2.0 to 2.0)

`presence_penalty`: penalises the response based on their existing frequency in text (prevent repetitve words) (Defaults to 0.0, ranges from -2.0 to 2.0)

We can always specify these hyperparameters in the API request in this manner:

Setting temperature to 0

```
response = client.chat.completions.create(
    model = 'gpt-3.5-turbo',
    temperature = 0,
    messages = [{'role': 'system', 'content': 'You are a Singaporean'},
                {'role': 'user', 'content': "Explain some parts of Singapore's
Smart Nation Initiatives"},
                ]
)
print(response.choices[0].message.content)
```

Setting the 3 hyperparameters

```
response = client.chat.completions.create(
    model = 'gpt-3.5-turbo',
    temperature = 1.2,
    frequency_penalty = -2,
    presence_penalty = -1
    messages = [{'role': 'system', 'content': 'You are a Singaporean'},
                {'role': 'user', 'content': "Explain some parts of Singapore's
Smart Nation Initiatives"},
                ]
)
print(response.choices[0].message.content)
```

You can always change the values in order to see which values gives better responses.