COMP 10261: The OpenAI Text Completion API

Sam Scott, Mohawk College, August 2023

Previously you learned some of the details of how **Large Language Models (LLMs)** generate text. This week, you'll be looking at how to embed GPT functionality into an app using the OpenAI API.

In this handout, the focus is on **Text Completion** – you provide a **prompt** and the **model** generates text that it thinks is likely to follow that prompt. With careful **Prompt Engineering**, you can use GPT models to implement many different types of functionality.

The Cost of the OpenAI API

From this point forward you should be prepared to spend a little bit of your own money. It's possible that the free credit you got when you signed up for OpenAI will be all you need for this course, but if not, you will have to provide credit card information to keep your experiments running. OpenAI charges by the token, so as long as you are not releasing your creations into the wild, you should not need to spend very much.

If you run out of free credits, go to https://platform.openai.com/account, then **Billing**. You can add a **Payment Method** here, and you should also set **Usage Limits**. I set a $1 soft and $2 hard limit and I have yet to hit the soft limit for normal usage when the course is running. But just in case you end up using the API more than I do, you should be ready to spend up to $30 if necessary. It's a small price to pay for what you're learning and it's the only extra cost for this course.

Free Alternatives

If you are dead set against spending money, there are freely-available large language models out there. For example, GPT-2 models of various sizes are available at https://huggingface.co/gpt2 and are not too difficult to use (though you must install the **transformers** package using **pip** first).

But there are two big drawbacks to the free models:

1. They are nowhere near as good as GPT-3 and GPT-4.
2. You will have to run the models on your own machine. For the smallest version of GPT-2, this may work, but for the larger and more capable versions of the model, the processing required may be too much to make this solution feasible.

USING THE TEXT COMPLETION API

The **gpt3example.py** file is a simple example of how to use the **openai** library for **Text Completion**. To run this program, you will need to install the **openai** module:

```
pip install openai
```

You will also need an **api key**. You can get one by registering at https://platform.openai.com/ if you have not already done so. Go to **View API Keys** and then **Create New Secret Key**. Make sure you copy it once its generated because you only get to see it once.

Setting Up

Once you have your key and the **openai** module, you can import and set your key like this:

```
import openai
openai.api_key = "<key goes here>"
```

Text Completion

In **gpt3example.py**, Each user utterance is sent as a prompt, as shown below:

```
response = openai.Completion.create(engine="text-davinci-003",
prompt=question, max_tokens=64, temperature=0.7)
```

The parameters you set here are the same ones that you can explore in the OpenAI playground.

You must specify the **engine** to be used. In the playground, this is referred to as the **model**. When you select models in the playground you also get a little bit of information about the cost and speed of each one. It might be a good idea to use the cheapest models during development and then switch to more expensive models in production. For more information on pricing, see https://openai.com/pricing.

The **prompt** is the other necessary parameter – it's just the text that you want completed.

Other parameters include the limit (**max_tokens**) on how much text should be produced and the **temperature**, which sets how random the response should be. If you set the **temperature** higher, you'll get different responses to the same prompt because it will not always choose the token with the highest probability at each step. If you set it to 0, the same prompt will always generate the same response.

For a full list of API parameters: https://platform.openai.com/docs/api-reference/completions/create

The Response

The **response** from GPT-3 is a **dictionary**-like object that contains a **choices** key with a **list** of possible responses (always containing one response unless you use a parameter to ask for more options). Each choice is also a dictionary with the **text** key storing the response itself. So the expression **response["choices"][0]["text"]** is the text of the first possible response.

The full structure of a typical response is shown below:

```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "logprobs": null,
      "text": "\n\nWelcome to our website. Our mission is to
      promote exploration and education through providing
      information and resources about geological and
      archaeological sites around the area."
    }
  ],
  "created": 1646056327,
  "id": "cmpl-4gajX7d4AMGColWYNTcKMh1AWjOSj",
  "model": "text-davinci:001",
  "object": "text_completion"
}
```

The sample bot in **gpt3example.py** just echoes this first choice to the user. It's pretty impressive just on its own, but to get the most out of the model for a particular task, or to get the bot to remember aspects of the conversation, you must engage in **Prompt Engineering**.

From a programming point of view, Prompt Engineering just means adding some text to what the user typed, or inserting what the user typed into a template, then sending the entire resulting text to the API. See **pirate_gpt3.py** for a simple example.

For more info about the text completion API, see the documentation at https://platform.openai.com/docs/guides/completion.

EXERCISES

1. Choose a prompt engineering strategy – either one of the strategies you implemented in the exercises last week, or one of the examples at https://platform.openai.com/examples – and build a simple bot based on it. You can use **pirate_gpt3.py** as a starter. Be sure to greet the user when the program runs and let them know what the bot is for and how they should use it.

   Test your bot thoroughly to see how well it deals with unexpected inputs. Can you "jailbreak" it and get it to exhibit undesirable behavior? If so, can you tweak the prompt a little bit to make it perform better in those circumstances?

2. Now try to add some dialog management through prompt engineering. You can use **gpt3example.py**, **pirate_gpt3.py**, or your response to question 1 as a starting point. To do dialog management means keeping a record of the conversation so far. For example, you could build a string that contains all the questions and responses so far, perhaps with "me:" and "you:" preceding each utterance, like this:

   ```
   me: Hello!
   you: Hi! How are you?
   me: I'm good, thanks. What's your name?
   you: My name is John. Nice to meet you.
   me: Nice to meet you, John. I'm Sam.
   you: Nice to meet you, Sam. What can I do for you?
   me: Well, tell me a bit about yourself.
   you:
   ```

   In the above example, the user has just typed, "Well, tell me a bit about yourself." The prompt provides context from the conversation so far and then cues GPT-3 to produce a response.

   This can get expensive, though, since OpenAI charges for both input and output tokens. So, it can be a good idea to limit the number of tokens you send.

   Here are some ideas on how to do that:

   a. As the conversation continues, you could exclude some of the earlier context. This is a trade-off. The model will not be constrained by things that were said far back in the conversation, but it will also will "forget" things that were said at the beginning.

   b. You could use GPT (perhaps a different model) to produce a summary of the conversation so far, then instead of presenting an entire transcript, present the summary in the prompt along with the most recent statement from the user and instructions on how the bot should respond.

3. Create a fine-tuned model for a specific task. For example, you could start from the "factual answering" example on the OpenAI website: https://platform.openai.com/examples/default-factual-answering. There are 11 question/answer pairs there that you could use as prompts and completions. You should add a few more of your own. Then attempt to create a fine-tune based on this data. It will only cost a few cents since you have very little data. You also might find the results a bit disappointing for the same reason. You could try adding stop sequences, but the best solution is to get more data!