day3

January 12, 2025

1 Day 3 - Conversational AI - aka Chatbot!

[1]:
```
# imports

import os
from dotenv import load_dotenv
from openai import OpenAI
import gradio as gr
```

[2]:
```
# Load environment variables in a file called .env
# Print the key prefixes to help with any debugging

load_dotenv()
openai_api_key = os.getenv('OPENAI_API_KEY')
anthropic_api_key = os.getenv('ANTHROPIC_API_KEY')
google_api_key = os.getenv('GOOGLE_API_KEY')

if openai_api_key:
    print(f"OpenAI API Key exists and begins {openai_api_key[:8]}")
else:
    print("OpenAI API Key not set")

if anthropic_api_key:
    print(f"Anthropic API Key exists and begins {anthropic_api_key[:7]}")
else:
    print("Anthropic API Key not set")

if google_api_key:
    print(f"Google API Key exists and begins {google_api_key[:8]}")
else:
    print("Google API Key not set")
```

```
OpenAI API Key exists and begins sk-proj-
Anthropic API Key exists and begins sk-ant-
Google API Key exists and begins AIzaSyDn
```

[3]:
```
# Initialize
```

```
openai = OpenAI()
MODEL = 'gpt-4o-mini'
```

[4]:
```
system_message = "You are a helpful assistant"
```

2 Please read this! A change from the video:

In the video, I explain how we now need to write a function called:

`chat(message, history)`

Which expects to receive `history` in a particular format, which we need to map to the OpenAI format before we call OpenAI:

```
[
    {"role": "system", "content": "system message here"},
    {"role": "user", "content": "first user prompt here"},
    {"role": "assistant", "content": "the assistant's response"},
    {"role": "user", "content": "the new user prompt"},
]
```

But Gradio has been upgraded! Now it will pass in `history` in the exact OpenAI format, perfect for us to send straight to OpenAI.

So our work just got easier!

We will write a function `chat(message, history)` where:
message is the prompt to use
history is the past conversation, in OpenAI format

We will combine the system message, history and latest message, then call OpenAI.

[5]:
```
# Simpler than in my video - we can easily create this function that calls␣
 ↪OpenAI
# It's now just 1 line of code to prepare the input to OpenAI!

def chat(message, history):
    messages = [{"role": "system", "content": system_message}] + history +␣
 ↪[{"role": "user", "content": message}]

    print("History is:")
    print(history)
    print("And messages is:")
    print(messages)

    stream = openai.chat.completions.create(model=MODEL, messages=messages,␣
 ↪stream=True)

    response = ""
    for chunk in stream:
```

```
        response += chunk.choices[0].delta.content or ''
        yield response
```

2.1 And then enter Gradio's magic!

[6]:
```
gr.ChatInterface(fn=chat, type="messages").launch()
```

```
* Running on local URL:  http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.
```

<IPython.core.display.HTML object>

[6]:

```
History is:
[]
And messages is:
[{'role': 'system', 'content': 'You are a helpful assistant'}, {'role': 'user',
'content': 'Hello'}]
History is:
[{'role': 'user', 'metadata': {'title': None}, 'content': 'Hello', 'options':
None}, {'role': 'assistant', 'metadata': {'title': None}, 'content': 'Hello! How
can I assist you today?', 'options': None}]
And messages is:
[{'role': 'system', 'content': 'You are a helpful assistant'}, {'role': 'user',
'metadata': {'title': None}, 'content': 'Hello', 'options': None}, {'role':
'assistant', 'metadata': {'title': None}, 'content': 'Hello! How can I assist
you today?', 'options': None}, {'role': 'user', 'content': "What's the weather
like in Helsinki right now?"}]
```

[7]:
```
system_message = "You are a helpful assistant in a clothes store. You should␣
 ↪try to gently encourage \
the customer to try items that are on sale. Hats are 60% off, and most other␣
 ↪items are 50% off. \
For example, if the customer says 'I'm looking to buy a hat', \
you could reply something like, 'Wonderful - we have lots of hats - including␣
 ↪several that are part of our sales evemt.'\
Encourage the customer to buy hats if they are unsure what to get."
```

[8]:
```
def chat(message, history):
    messages = [{"role": "system", "content": system_message}] + history +␣
 ↪[{"role": "user", "content": message}]

    stream = openai.chat.completions.create(model=MODEL, messages=messages,␣
 ↪stream=True)

    response = ""
```

```
    for chunk in stream:
        response += chunk.choices[0].delta.content or ''
        yield response
```

[9]:
```
gr.ChatInterface(fn=chat, type="messages").launch()
```

```
* Running on local URL:  http://127.0.0.1:7861

To create a public link, set `share=True` in `launch()`.
```

<IPython.core.display.HTML object>

[9]:

[10]:
```
system_message += "\nIf the customer asks for shoes, you should respond that␣
 ↪shoes are not on sale today, \
but remind the customer to look at hats!"
```

[11]:
```
gr.ChatInterface(fn=chat, type="messages").launch()
```

```
* Running on local URL:  http://127.0.0.1:7862

To create a public link, set `share=True` in `launch()`.
```

<IPython.core.display.HTML object>

[11]:

[12]:
```
# Fixed a bug in this function brilliantly identified by student Gabor M.!
# I've also improved the structure of this function

def chat(message, history):

    relevant_system_message = system_message
    if 'belt' in message:
        relevant_system_message += " The store does not sell belts; if you are␣
 ↪asked for belts, be sure to point out other items on sale."

    messages = [{"role": "system", "content": relevant_system_message}] +␣
 ↪history + [{"role": "user", "content": message}]

    stream = openai.chat.completions.create(model=MODEL, messages=messages,␣
 ↪stream=True)

    response = ""
    for chunk in stream:
        response += chunk.choices[0].delta.content or ''
        yield response
```

[13]:
```
gr.ChatInterface(fn=chat, type="messages").launch()
```

```
* Running on local URL:  http://127.0.0.1:7863

To create a public link, set `share=True` in `launch()`.

<IPython.core.display.HTML object>
```

[13]:

Business Applications

Conversational Assistants are of course a hugely common use case for Gen AI, and the latest frontier models are remarkably good at nuanced conversation. And Gradio makes it easy to have a user interface. Another crucial skill we covered is how to use prompting to provide context, information and examples. Consider how you could apply an AI Assistant to your business, and make yourself a prototype. Use the system prompt to give context on your business, and set the tone for the LLM.

[ ]: