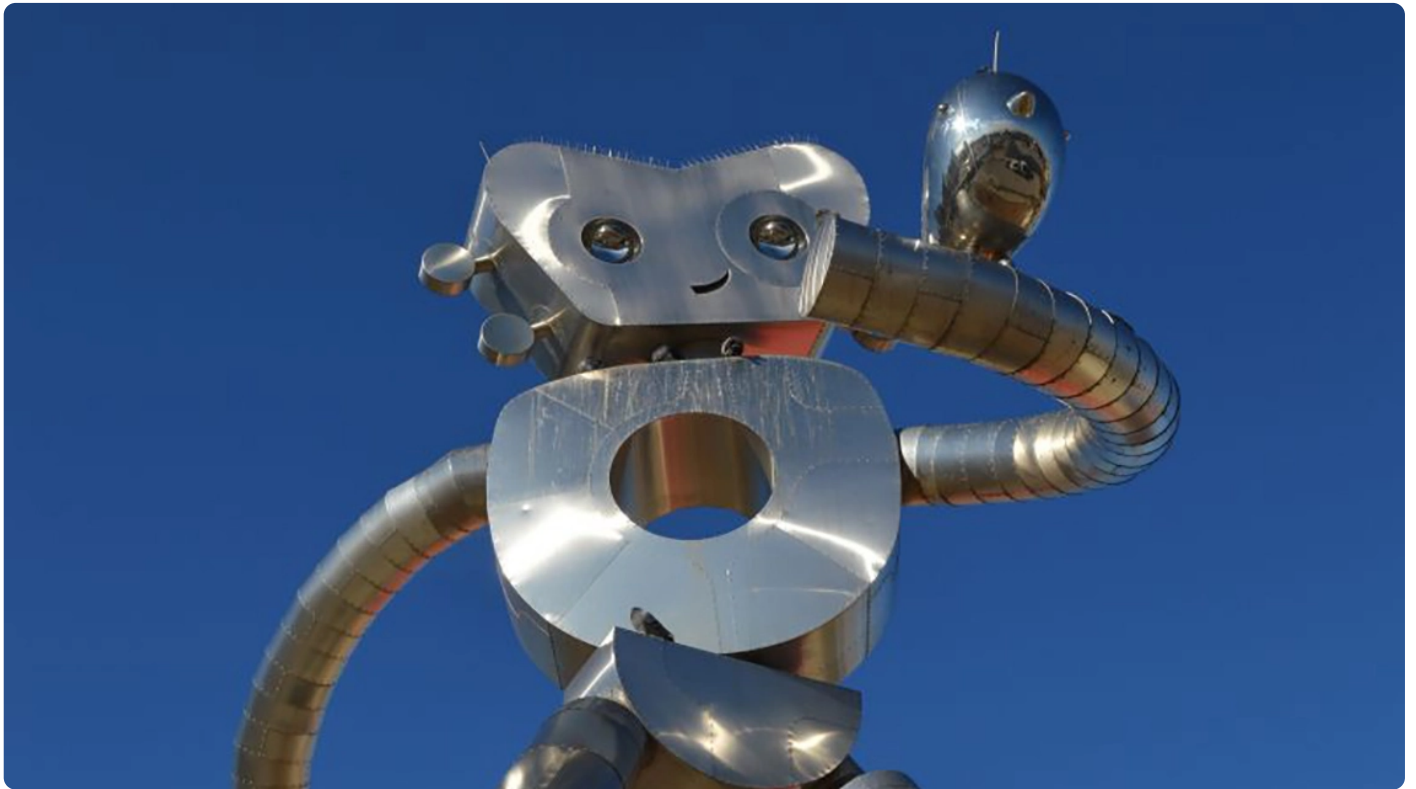


18 | 流式生成与模型微调，打造极致的对话体验

2023-04-17 徐文浩 来自北京

《AI大模型之美》



你好，我是徐文浩。

在之前介绍 llama-index 和 LangChain 的几讲里面，我们学习了如何将大语言模型和你自己的知识库组合到一起解决问题。这个方法中，我们不需要对我们使用的模型做任何调整，而是通过将我们的数据用 Embedding 向量索引起来，然后在使用的时候查询索引来解决问题。

不过，其实我们也完全可以利用我们自己的数据，创建一个新的模型来回答问题。这个方法，就是 OpenAI 提供的模型微调（Fine-tune）功能。这也是我们要探讨的大语言模型的最后一个主题。


如何进行模型微调？

模型微调，是因为无论是 ChatGPT 还是 GPT-4 都不是全知全能的 AI。在很多垂直的领域，它的回答还是常常会出错。其中很大一部分原因，是它也缺少特定领域的训练数据。而如果我

们有比较丰富的垂直领域的的数据，那么就可以利用这些数据来“微调”一个特别擅长这个垂直领域的模型。在这个模型“微调”完成之后，我们就可以直接向模型提问了。而不用再像之前使用 llama-index 或者 LangChain 那样，先通过 Embedding 来查询相关资料，然后把查找到的资料也一并提交给 OpenAI 来获得所需要的答案。

OpenAI 模型微调的过程，并不复杂。你只需要把数据提供给 OpenAI 就好了，对应的整个微调的过程是在云端的“黑盒子”里进行的。需要提供的数据格式是一个文本文件，每一行都是一个 Prompt，以及对应这个 Prompt 的 Completion 接口会生成的内容。

就像下面的示例：

 复制代码

```
1 {"prompt": "<prompt text>", "completion": "<ideal generated text>"}
2 {"prompt": "<prompt text>", "completion": "<ideal generated text>"}
3 {"prompt": "<prompt text>", "completion": "<ideal generated text>"}
4 ...
```


模型微调的过程，就是根据输入的内容，在原来的基础模型上训练。这个基础模型，就是我们 [第 8 讲](#) 介绍过的 Ada、Babbage、Curie 和 Davinci 其中的一个。每一个示例，都会导致基础模型原有参数发生变化。整个微调过程结束之后，变化后的参数就会被固定下来，变成一个只有你可以使用的新模型。

如果你提供了很多医疗行业的文本内容，那么微调出来的新模型就会拥有更多医疗领域的知识，以及对话的风格。而如果你给的是笑话大全，那么微调出来的模型就更擅长讲笑话。而且要注意，微调之后的模型，不仅有你用来微调的数据的相关知识，原先基础模型里面的绝大部分知识和能力它也还都保留着。

来一个擅长写“历史英雄人物和奥特曼一起打怪兽”的 AI

那今天我们来微调一个什么样的模型呢？我周围有不少朋友家里都有孩子，都特别迷恋奥特曼打怪兽的故事。他们就向我提过一个需求，说能不能利用 ChatGPT 来做一个专门讲奥特曼打怪兽故事的应用。可以是可以，不过，为了让这个故事既能精彩一点，又有点教育意义，我们

就再找一些历史上的英雄人物，赋予他们一些超能力，来和奥特曼一起打怪兽。而对应的故事数据，我们也用 ChatGPT 的模型来帮我们生成。

 复制代码

```
1 import os,openai,backoff
2 import pandas as pd
3
4 openai.api_key = os.getenv("OPENAI_API_KEY")
5 dynasties= ['唐', '宋', '元', '明', '清', '汉', '魏', '晋', '南北朝']
6 super_powers = ['隐形', '飞行', '读心术', '瞬间移动', '不死之身', '喷火']
7 story_types = ['轻松', '努力', '艰难']
8
9 @backoff.on_exception(backoff.expo, openai.error.RateLimitError)
10 def gpt35(prompt, max_tokens=2048, temperature=0.5, top_p=1, frequency_penalty=0,
11         response = openai.Completion.create(
12             engine="text-davinci-003",
13             prompt=prompt,
14             max_tokens=max_tokens,
15             temperature=temperature,
16             top_p=top_p,
17             frequency_penalty=frequency_penalty,
18             presence_penalty=presence_penalty)
19     return response["choices"][0]["text"]
20
21 def prepare_stories(dynasties, super_powers, story_types, output_file="data/ultra
22 df = pd.DataFrame()
23 repeat = 3
24 for dynasty in dynasties:
25     for super_power in super_powers:
26         for story_type in story_types:
27             for i in range(repeat):
28                 prompt = f"""请你用中文写一段300字的故事，情节跌宕起伏，讲述一位{
29                 story = gpt35(prompt)
30                 row = {"dynasty": dynasty, "super_power": super_power, "s
31                 row = pd.DataFrame([row])
32                 df = pd.concat([df, row], axis=0, ignore_index=True)
33
34     df.to_csv("data/ultraman_stories.csv")
35
36 prepare_stories(dynasties, super_powers, story_types)
```

这部分代码非常简单，我们定义了一系列朝代、超能力和故事的类型。然后通过三重循环，让 AI 根据这三者的组合来生成一系列故事。这些生成出来的故事，也就构成了我们用来微调模


型的训练数据。因为数据量不大，我就直接用 CSV 把它存下来了。在这个过程中，数据是一条条生成的，比较慢，也比较消耗 Token，你可以不用运行，直接拿我运行后生成的结果数据就好。

拿到了这些数据，我们就可以来微调模型了。我们之前已经通过 pip 安装了 OpenAI 的包，这里面自带了命令行工具，方便我们把对应的 CSV 格式的数据转换成微调模型所需要的 JSONL 格式的文件。

 复制代码

```
1 df = pd.read_csv("data/ultraman_stories.csv")
2 df['sub_prompt'] = df['dynasty'] + "," + df['super_power'] + "," + df['story_type']
3 prepared_data = df.loc[:,['sub_prompt','story']]
4 prepared_data.rename(columns={'sub_prompt':'prompt', 'story':'completion'}, inplace=True)
5 prepared_data.to_csv('data/prepared_data.csv',index=False)
6
7 import subprocess
8
9 subprocess.run('openai tools fine_tunes.prepare_data --file data/prepared_data.csv')
```

输出结果：


 复制代码

```
1 .....
2 Wrote modified file to `data/prepared_data_prepared.jsonl`
3 Feel free to take a look!
4 Now use that file when fine-tuning:
5 > openai api fine_tunes.create -t "data/prepared_data_prepared.jsonl"
6 After you've fine-tuned a model, remember that your prompt has to end with the in
7 Once your model starts training, it'll approximately take 8.82 minutes to train a
8
9 CompletedProcess(args=['openai', 'tools', 'fine_tunes.prepare_data', '--file', 'd
```

上面的代码主要做了两个动作。首先，是对数据做了一些处理，来准备微调。对于微调，我们使用的 Prompt 不再是一个完整的句子，而是只用了“朝代” + “超能力” + “故事类型”拼接在一起的字符串，中间用逗号隔开。然后把这个字符串和生成的故事，用 Prompt 和 Completion 作为列名存储成了一个 CSV。

其次，我们通过 subprocess 调用了命令行里的 OpenAI 工具，把上面的 CSV 文件，转化成了一个 JSONL 格式的文件。从输出的日志里面可以看到，这个文件叫做 data/prepared_data_prepared.jsonl。


如果我们打开这个 JSONL 文件看一眼，是下面这样的。

 复制代码

```
1 {"prompt":"唐,隐形,轻松 ->","completion":" \n\n一位叫做李明的英雄人物，出生在唐朝时期。他  
2 {"prompt":"唐,隐形,轻松 ->","completion":" \n\n这是一个关于英雄的故事，发生在唐朝时期的中
```


可以看到，转换后的数据文件，在 Prompt 的最后，多了一个 “->” 符号。而在 Completion 的开头，多了两个 “\n\n” 的换行，结尾则是多了一个 “。”。这是为了方便我们后续在使用这个模型生成数据的时候，控制生成结果。未来在使用模型的时候，Prompt 需要以 “->\n” 这个提示符结束，并且将 stop 设置成 “。”。这样，模型就会自然套用我们微调里的模式来生成文本。

有了准备好的数据，我们只要再通过 subprocess 调用 OpenAI 的命令行工具，来提交微调的指令就可以了。

 复制代码

```
1 subprocess.run('openai api fine_tunes.create --training_file data/prepared_data_p
```


输出结果：

 复制代码

```
1 Upload progress: 100%|██████████| 446k/446k [00:00<00:00, 201Mit/s]  
2 Uploaded file from data/prepared_data_prepared.jsonl: file-yn0BfnPmgvf7n0sfQzQRbb  
3 Created fine-tune: ft-3oxkr1zBVB4fJWogJDDjQbr0  
4 Streaming events until fine-tuning is complete...  
5 (Ctrl-C will interrupt the stream, but not cancel the fine-tune)  
6 [2023-04-04 10:51:51] Created fine-tune: ft-3oxkr1zBVB4fJWogJDDjQbr0  
7  
8 CompletedProcess(args=['openai', 'api', 'fine_tunes.create', '--training_file', 'data/prepared_data_prepared.jsonl', 'ft-3oxkr1zBVB4fJWogJDDjQbr0'])
```


在这个微调的指令里面，我们指定了三个参数，分别是用来训练的数据文件、一个基础模型，以及生成模型的后缀。这里，我们选用了 Curie 作为基础模型，因为是讲奥特曼的故事，所以模型后缀我给它取了一个 ultraman 的名字。

我们的数据量不大，所以微调很快，几分钟就能完成。那接下来我们就可以使用这个模型了。我们可以通过下面的 `fine_tunes.list` 指令，找出所有我们微调的模型。

 复制代码

```
1 subprocess.run('openai api fine_tunes.list'.split())
```

输出结果：


 复制代码

```
1 {
2   "data": [
3     {
4       "created_at": 1680576711,
5       "fine_tuned_model": "curie:ft-bothub-ai:ultraman-2023-04-04-03-03-26",
6       "hyperparams": {
7         "batch_size": 1,
8         "learning_rate_multiplier": 0.2,
9         "n_epochs": 4,
10        "prompt_loss_weight": 0.01
11      },
12      "id": "ft-3oxkr1zBVB4fJWogJDDjQbr0",
13      "model": "curie",
14      "object": "fine-tune",
15      "organization_id": "YOUR_ORGANIZATION_ID",
16      "result_files": [
17        {
18          "bytes": 107785,
19          "created_at": 1680577408,
20          "filename": "compiled_results.csv",
21          "id": "RESULT_FILE_ID",
22          "object": "file",
23          "purpose": "fine-tune-results",
24          "status": "processed",
25          "status_details": null
26        }
27      ]
28    },
29  ]
30 }
```

```
29     "object": "list"
30 }
31 CompletedProcess(args=['openai', 'api', 'fine_tunes.list'], returncode=0)
```


在输出的 JSON 里面，你可以看到我们有一个 `fine_tuned_model` 字段，里面的值叫做 `"curie:ft-bothub-ai:ultraman-2023-04-04-03-03-26"`，这个就是刚刚让 OpenAI 给我们微调完的模型。

这个模型的使用方法，和我们使用 `text-davinci-003` 之类的模型是一样的，只要在 API 里面把对应的 `model` 字段换掉就好了，对应的代码我也放在了下面。

 复制代码

```
1 import os
2 import openai
3
4 openai.api_key = os.getenv("OPENAI_API_KEY")
5
6 def write_a_story(prompt):
7     response = openai.Completion.create(
8         model="curie:ft-bothub-ai:ultraman-2023-04-04-03-03-26",
9         prompt=prompt,
10        temperature=0.7,
11        max_tokens=2000,
12        top_p=1,
13        stop=["."])
14    return response["choices"][0]["text"]
15
16 story = write_a_story("宋,发射激光,艰难 ->\n")
17 print(story)
```

输出结果：

 复制代码

```
1 宋朝时期，有一位叫林先生的英雄人物，他勇敢而又坚韧，曾经拯救过无数的人民，他的英勇表现让他赢得了
2 一天，林先生突然发现自己穿越到了现代，他发现自己拥有了一种神奇的超能力，可以发射激光，他开始研究
3 林先生决定把自己的能力用来拯救人类，于是他和奥特曼一起出发，开始与怪兽作战。他们一路走来，林先生
4 在一场艰苦的战斗中，林先生和奥特曼终于击败了怪兽，拯救了人类。林先生也因此获得了无数的赞誉，他也
5 林先生的故事被传唱了几百年，他的英勇事迹也成为了一个永恒的传奇，让人们永远不忘。
```



对应在调用模型的时候，我们使用的提示语就是“朝代” + “超能力” + “故事类型”，并且跟着“->\n”，而 stop 则是设置成了“。”。

因为这是一个微调的模型，它不仅拥有我们训练数据提供的知识，也包括基础模型里的各种信息。所以我们使用的朝代、超能力和故事类型也可以是在之前微调数据里面没有出现过的。比如，上面的例子里，我们使用的超能力叫做“发射激光”，并不是我们拿来微调的数据里面有一种超能力。你可以试试看，使用别的朝代、故事的类型，效果会是怎么样的。

 复制代码

```
1 story = write_a_story("秦,龙卷风,辛苦 ->\n")
2 print(story)
```

输出结果：

 复制代码

```
1
2 曾经有一位叫苏轼的英雄人物，他曾经英勇地抵抗过许多强大的敌人，拯救了许多被危险封印的百姓。他曾经在
3 然而，苏轼却在一次激烈的战斗中牺牲了，他的灵魂被封印在一个古老的石头里，隔着一层玻璃，一直沉睡了1
4 苏轼的灵魂在穿越时空，来到了现代，他发现自己拥有了一种超能力，这就是龙卷风，他可以使自己的身体具
5 苏轼在现代的世界里，发现了一种可怕的怪兽，它们正在摧毁着人类的家园，苏轼决定要拯救这个世界，于是
6 在苏轼和奥特曼的帮助下，苏轼利用自己的超能力，一次次击退怪兽的攻击，最终他们成功地打败了怪兽，拯
7 苏轼的事迹在这里传唱了很久，他成为了一位永恒的英雄，他的故事也被传唱了下来，让人们永远不会忘记他的
```

模型微调的成本考量

细心的人可能注意到了，我们这里选用的基础模型是 Curie，而不是效果最好的 Davinci。之所以做出这样的选择，是出于成本的考虑。

模型	训练成本	使用成本
Ada	\$0.0004 / 1K Tokens	\$0.0016 / 1K Tokens
Babbage	\$0.0006 / 1K Tokens	\$0.0024 / 1K Tokens
Curie	\$0.0030 / 1K Tokens	\$0.0120 / 1K Tokens
Davinci	\$0.0300 / 1K Tokens	\$0.1200 / 1K Tokens



注：数据来源于 <https://openai.com/pricing#language-models>

使用微调模型的成本要远远高于使用 OpenAI 内置的模型。以 Davinci 为基础微调的模型，使用的时候，每 1000 个 Token 的成本是 0.12 美元，是使用内置的 text-davinci-003 的 6 倍，是我们最常用的 gpt-3.5-turbo 的 60 倍。所以，如果只是一般的讲故事的应用，这个成本实在是太高了。就算是我们选择基于 Curie 微调，1000 个 Token 的使用成本也在 0.012 美元，虽然比 text-davinci-003 要便宜，但也是 gpt-3.5-turbo 的 6 倍。

对于模型微调的效果，我们也可以通过一个 OpenAI 提供的命令 `fine_tunes.results` 来看。对应的，我们需要提供给它一个微调任务的 id。这个 id，可以在 `fine_tunes.list` 列出的 `fine_tunes` 模型的 id 参数里找到。

复制代码

```
1 subprocess.run('openai api fine_tunes.results -i ft-3oxkr1zBVB4fJWogJDDjQbr0'.spl
```

输出结果：

复制代码

```
1 step,elapsed_tokens,elapsed_examples,training_loss,training_sequence_accuracy,tra
2 1,625,1,0.8805545861742778,0.0,0.75
3 2,1258,2,0.8059815050491868,0.0,0.7766830870279147
4 3,1859,3,0.7964038042175758,0.0,0.7862068965517242
```

```
5 4,2548,4,0.805052303553852,0.0,0.7774436090225564
6 5,3197,5,0.7503930440556053,0.0,0.7808
7 6,3846,6,0.7992317049403261,0.0,0.7770700636942676
8 7,4775,7,0.6649006477473822,0.0,0.7927232635060639
9 8,5432,8,0.6493354803676822,0.0,0.8049921996879875
10 9,6265,9,0.6568901059838095,0.0,0.802937576499388
11 10,7122,10,0.6578856167468091,0.0,0.8100358422939068
12 11,7827,11,0.5687322367928961,0.0,0.8279411764705882
13 12,8404,12,0.6334827334911788,0.0,0.8172043010752689
14 13,9061,13,0.5771709139683721,0.0,0.825
15 14,9822,14,0.6079089517825593,0.0,0.8100407055630936
16 15,10399,15,0.6481047367374327,0.0,0.8154121863799283
17 16,11208,16,0.5528688982071029,0.0,0.8352490421455939
18 17,11913,17,0.6525803676480848,0.0,0.8093841642228738
19 18,12546,18,0.5230526420679229,0.0,0.8363047001620746
20 19,13163,19,0.6065665546680247,0.0,0.8236272878535774
21 20,13796,20,0.5983224045073889,0.0,0.8199672667757774
22 21,14549,21,0.6440337136896056,0.0,0.8267394270122783
23 22,15190,22,0.6029605409912032,0.0,0.8110749185667753
24 23,15759,23,0.5089513997451476,0.0,0.838475499092559
25 24,16440,24,0.557213810807506,0.0,0.8265460030165912
26 ...
27 1855,1228711,1855,0.2610049068084409,0.0,0.9219765929778934
28 1856,1229312,1856,0.21196416716076574,0.0,0.9312714776632303
29 1857,1229945,1857,0.14050147435694596,0.0,0.9556650246305419
```


在这个命令的输出结果里，你可以在第二列 `elapsed_tokens` 看到训练消耗的 Token 数量。而最后一列 `training_token_accuracy`，则代表微调后的模型，成功预测微调的数据里下一个 Token 的准确率。在我们使用的这个例子里面，可以看到一开始准确率只有 75%，但是随着训练数据迭代轮数的增加，准确率越来越高，达到了 95% 以上。

增量训练，不断优化模型

微调模型比较高昂的价格，限制了它的使用。**不过，微调模型还有一个能力，就是我们可以已经微调了的模型上根据新数据做进一步地微调。**这个在很多垂直领域是非常有用，比如在医学、金融这样的领域，我们就可以不断收集新的数据，不断在前一个微调模型的基础之上继续微调我们的模型，让模型的效果越来越好。而这些领域往往也能承受更高一些的成本。


进一步地微调其实操作起来并不复杂，就是再准备一些数据，以之前已经微调好的模型为基础模型来操作就好了。

生成一些额外的数据：

 复制代码


```
1 dynasties= ['秦', '五代', '隋']
2 super_powers = ['龙卷风', '冰冻大海', '流星火雨']
3 story_types = ['轻松', '努力', '艰难', '勇敢', '辛苦']
4
5 new_stories = "data/ultraman_stories_more.csv"
6 prepare_stories(dynasties, super_powers, story_types, repeat=3, output_file=new_s
```

转换数据：

 复制代码

```
1 df = pd.read_csv(new_stories)
2 df['sub_prompt'] = df['dynasty'] + "," + df['super_power'] + "," + df['story_type']
3 prepared_data = df.loc[:,['sub_prompt','story']]
4 prepared_data.rename(columns={'sub_prompt':'prompt', 'story':'completion'}, inplace=True)
5 new_stories_prepared = "data/prepared_data_more.csv"
6 prepared_data.to_csv(new_stories_prepared, index=False)
7
8 subprocess.run('openai tools fine_tunes.prepare_data --file data/prepared_data_mo
```

继续微调：


 复制代码

```
1 subprocess.run('openai api fine_tunes.create --training_file data/prepared_data_m
```

在原有的模型上微调的时候，我们要修改两个参数。


1. 第一个是 model 参数，我们把 Curie 换成了我们刚才微调之后的模型 **curie:ft-bothub-ai:ultraman-2023-04-04-03-03-26**。
2. 第二个是 learning_rate_multiplier，这个参数的默认值是根据你的样本数量在 0.05 到 0.2 不等。如果你继续微调的样本数要比之前微调的数据量小很多，你就可以调得大一点。

微调更新之后，模型的名称没有变，老的模型就被更新成了微调后的新模型，我们再来试一下这个新模型。

 复制代码

```
1 fine_tuned = write_a_story("五代,流星火雨,艰难 ->\n")
2 print(fine_tuned)
```


输出结果：

 复制代码

```
1 这是一个发生在一个古老的世界，一个叫做“六代”的世界。这个世界有着一种叫做“超能力”的特性，可以让人
2 一位叫做“英雄”的人物，他来自于六代，但他拥有了一种叫做“流星火雨”的超能力，他可以把自己的身体变成
3 他来到现代，发现这个世界变得越来越危险，有一种叫做“怪兽”的存在，他们想要毁灭这个世界。英雄决定帮
4 英雄凭借着自己的超能力，以及奥特曼的力量，战胜了怪兽，拯救了这个世界。最后，英雄又一次穿越回六代，
```

流式生成


通过模型微调，我们拥有了一个可以讲故事的 AI 模型。不过，故事生成的体验稍微有点差。它不像是我们在 ChatGPT 的 Web 界面里那样一个词一个词地蹦出来，就像一个真人在给你讲故事那样。不过要做到这一点也并不难，因为 OpenAI 的 Completion 接口是提供了这样返回结果的模式的，你只需要把代码小小地修改一下就好了。

 复制代码

```
1 def write_a_story_by_stream(prompt):
2     response = openai.Completion.create(
3         model="curie:ft-bothub-ai:ultraman-2023-04-04-03-03-26",
4         prompt=prompt,
5         temperature=0.7,
6         max_tokens=2000,
7         stream=True,
8         top_p=1,
9         stop=["."])
10    return response
11
12 response = write_a_story_by_stream("汉,冰冻大海,艰难 ->\n")
13
14 for event in response:
15     event_text = event['choices'][0]['text']
```

```
16 print(event_text, end = '')
```

输出结果：

 复制代码

- 1 一位叫李英的汉朝时期的英雄人物，穿越到了现代，拥有了一种超能力，可以把自己的身体冰冻到极限，他发现
- 2 李英发现，地球正面临着一个叫做怪兽的强大敌人的威胁，他决定去帮助奥特曼一起打败怪兽。于是，他和奥特曼
- 3 李英受到了所有人的赞赏，他也成为了一个英雄，他的事迹被传颂了几百年，他的故事也被记录在历史书中，他

我们在调用 Completion 接口的时候，启用了 stream=True 这个参数。然后对于返回结果，我们不再是直接拿到整个 response 然后打印出来。而是拿到一个可以通过迭代器访问的一系列 events，每一个 event 都包含了一部分新生成的文本。你试着运行一下这段代码，就能体验到 AI 把一个个词吐给你，好像真的在实时讲故事一样的感觉了。

小结

好了，今天的课程到这里也就结束了。这一讲里，我们一起学习了 OpenAI 大语言模型里的最后两个功能。

第一个是模型微调，模型微调给我们提供了一个非常实用的能力，**我们可以利用自己的数据，在 OpenAI 的基础模型上，调整模型参数生成一个新模型**。这样我们就能够根据自己专有的垂直领域的的数据，来生产一个专属于我们自己的模型。而且，我们可以根据新收集到的数据，不断在这个模型上继续微调迭代。不过，微调后的模型使用成本比较高，你需要自己核算一下，究竟是微调模型 ROI 比较高，还是使用前面的外部知识库的方式更划算一些。

在模型微调之外，我们还了解了 OpenAI 接口上的一个小功能，也就是**流式地数据生成**。通过开启流式地文本生成，我们可以交付给用户更好的交互体验。特别是在使用比较慢的模型，比如 GPT-4，或者生成的文本很长的時候，效果特别明显。用户不需要等上几十秒才能看到结果。

那到这里，整个课程的大语言模型部分我们也就介绍完了。从最基本的两个 API，Completion 和 Embedding 开始，我为你介绍了各种各样的应用场景和使用方法。可以看

到，现在的大语言模型几乎是“万能”的。下可以拿来做机器学习的输入数据，上可以直接让它自己决定调用什么 API，怎么解决用户的问题。相信看到这里的你，已经掌握如何使用大语言模型了，接下来就要多想想在你的实际工作里如何把它用起来了。

思考题

这是大语言模型部分的最后一讲，所以我就多给你留一些练习题。

1. 在这一讲生成数据的时候，我们一条条去生成故事特别慢，而且每个组合的故事都要生成三条，特别消耗 Token。你想想这部分的代码，如何根据之前学到的内容优化一下呢？
2. 你能不能尝试通过流式处理，做一个讲故事的小应用？并且在界面上，用户能够看到故事真的是一个词儿一个词儿地蹦出来的。
3. OpenAI 的模型微调，其实还有很多更丰富的用法，比如可以拿来做分类，或者命名实体的提取。你可以去官网的 [🔗 Specific Guidelines](#) 部分看一看，来试着微调一个模型，根据电商商品页的属性信息来写商品的详情描述。

欢迎你把你实现这个方法的功能，以及最后的效果体验分享到评论区，也欢迎你把这一讲分享给感兴趣的朋友，我们下一讲再见。

推荐阅读

OpenAI 在自己的 [🔗 官方文档](#)里，推荐了通过 Weight & Bias 这个公司的产品，来追踪微调后的模型的实验、模型与数据集。Weight & Bias 也在自己的 [🔗 文档](#)里，提供了一个对 WIT 数据集进行模型微调的 [🔗 Notebook](#)，你有兴趣的话也可以去看一下。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 (13)



Toni

2023-04-18 来自瑞士

模型的微调(GPT fine-tuning)本质上就是将 GPT 过于宽泛化的处理调窄到某个特殊的范围，而这个在GPT大环境下的特殊范围需要使用满足一定条件的'数据集'来打造，可能还需要反复打造。

读过并记住了全部牛津大字典，莎翁作品的 GPT，实力摆在那儿了，能聊，上知天文下晓地理，侃功盖世。但使用好 GPT 的超能力并不是一件容易事，尤其是当你并不满足于它的夸夸其谈，那么就微调吧。

微调容易吗？看怎么说了，用于微调的代码已经是简单到不能再简单了，不会编程照样上手，但要达到一特定目的，就不那么容易了。比如：让 GPT 用鲁迅的笔法来生成一篇短文来抨击环保中的问题。难在以鲁迅的口吻。

那就微调吧，喂给 GPT 大量鲁迅写的文章，让 GPT 再学习。本来想借老师这课讲故事的例子来试试将故事员变成鲁迅，但短时间内是无法完成了。

既然无法微调，那就试试 GPT 中的 Prompt，提示词的本质就是收窄范围，与 fine-tuning 有异曲同工之妙。用 OpenAI 的 Completion 在 engine="text-davinci-003" 的驱动下，生成了下面这段文例：

"垃圾分类，一种'新时尚'，连阿贵都在谈论它。每个人似乎都很努力，然而，在许多人的心中，它只不过是一种无聊的行为，他们只是为了显得出色而做。

只有少数真正的尝试改变者，多数人只是在虚情假意下，把它当成一种'潮流'。

追求'时尚'的'环保'是空洞的。与其虚情假意，不如从自身出发，努力去改变。"

还是太平淡，太啰嗦了。如何给出提示词使它更像鲁迅？但也许鲁迅喜欢："你们的白话文讲得比我好。"

在无法微调时就尝试一下 Prompt。



👍 5



金口

2023-04-18 来自广东

只要能加载微调后的模型，是不是就可以不用openai了？

作者回复：这一讲的微调后的模型还是在OpenAI的云端的。基于开源模型自己用GPU微调，才能不使

用openai。

共 2 条评论 >

👍 3



Oli张帆

2023-04-18 来自北京

从成本的角度看，微调是不是更适合用来做意图分析的分类任务？我这样想的理由是，因为分类任务相对简单可以用比较便宜的模型来微调，第二，训练好之后，每次调用不必发大量context的提示，这样消耗的token会少很多。请老师指正。

作者回复：我个人主观的感觉是，更适合垂直领域的应用。比如你有大量的法律类的文书、金融类的文书等等，单独微调一个模型给自己用。

能够承受相对高成本，并且有大量垂直语料的应用看起来比较合适。

只是意图分析，我觉得不太需要微调语言生成模型，自己训练一个分类模型就好了。



👍 2



Geek_4ec46c

2023-04-17 来自中国台湾

老师,问下,Langchain的流式生成似乎不支持迭代的显示方式,好像是通过callback来实现,但是这样在开发web的时候似乎就没办法做成应用了

作者回复：Callback一样可以啊。你在前端和后端建立一个Websocket，然后后端拿到Callback之后往前端Push消息就可以了。

共 2 条评论 >

👍 1



ACK

2023-04-17 来自中国香港

我家孩子喜欢听恐龙大战奥特曼的故事。

用 GPT-4 来直接生成吧，故事太老套，每次生成的情节很相近。还要人工来润色。

共 1 条评论 >

👍 1



朱朱

2023-05-16 来自浙江

老师，请教下，微调之后的模型在 openai 的云端会存在多久，我直接用您微调后的模型，提示无法找到了

作者回复: 微调后的模型，不删除一直会在云端。但是每个人的模型是隔离的呀，你肯定是调用不了我微调的模型的。你需要自己微调一个才能调用。

共 2 条评论 >



蔡雪钧

2023-05-08 来自北京

如果用 A, B, C三个垂直领域的数据做模型微调，那么微调后的模型是同时增强了A, B, C三块的能力么？比如把数理化的题喂给大模型，后面大模型是不是数理化整体都会变强？

作者回复: 是的，但是也有overfit过拟合导致其他方面能力变弱的风险。



weiwei

2023-04-20 来自浙江

徐老师，我们AI部门主要做“异常数据检测”业务。就是离线把金融的异常数据捕获出来。之前这块是用规则引擎+人工审核来做的，人力成本挺大。

这块业务，理论上能用fine tuning模型来做吗？

作者回复: 这个有专门的策略和算法啊，可以看看 anomaly detector 类型的策略和算法。

可以看看kaggle比赛里面大家的解决方案找找思路？ <https://www.kaggle.com/ealaxi/paysim1>



stg609

2023-04-19 来自浙江

stop到底是怎么发挥作用的？一遇到stop的字符就停止生成了？如果我们喂给它的csv数据中就有很多. 会怎么样？

作者回复: 一遇到就停止生成

你可以试一下，那样的话，预处理过程生成的 stop word 会变



莱森

2023-04-18 来自四川

可以请教一下老师如果用了链式调用，怎样才能更好地实现流式生成呢？

作者回复：可以参看官方文档关于 Stream Response的支持

<https://python.langchain.com/en/latest/modules/models/chat/examples/streaming.html>



Warren

2023-04-17 来自广东

在这一讲生成数据的时候，我们一条条去生成故事特别慢，而且每个组合的故事都要生成三条，特别消耗 Token。你想想这部分的代码，如何根据之前学到的内容优化一下呢？是不是可以把completion接口的n设为3，一次返回3个故事，可以减少提交的token？

作者回复：这是一种方式，还有更多的办法。



树静风止

2023-04-17 来自北京

面向模型编程



zhihai.tu

2023-04-17 来自上海

能不能联动tts，把故事实时播报出来。

作者回复：当然可以啊，后面20讲会讲解如何接入TTS



