3

3.1 模型输入输出

LangChain 打通了大语言模型应用开发的“最后一公里”。

任何大语言模型应用的核心元素是 —— 模型的输入 Input 和输出 Output (简称模型输入输出 (Model I/O),下面都用 I 表示模型的输入 Input，O 表示模型的输出 Output）。

最近一年左右，通过强化学习等方法对模型进行微调（Fine-tuning），以及在提示（Prompt）中使用各种方法，以从这些模型中获得最佳效果。一旦我们能够做到这一点，从这些大型语言模型中我们可以得到一些非常好的结果。问题是，它们仍然无法以接口方式访问现有数据和API,无法与真实的世界连接起来就很难应用,尤其是复杂的应用。

如果我们想要使用这些模型构建一个应用程序，我们需要某种方式在大型语言模型和传统软件之间进行接口交互。同时，越来越多的大语言模型涌现出来，模型输入输出变得异常重要。LangChain 为我们提供了与任何大型语言模型进行交互的基本组件。

LangChain 不仅可以方便地与最新、最强大的模型如GPT-4进行交互，还可以与本地私有化部署的语言模型，或是在 HuggingFace 上找到的开源模型进行交互。

只需几行代码，就可以实现与这些模型进行对话。如何使用 Langchain 的基础组件模型输入输出来访问大型语言模型？来看看下面这张示意图：

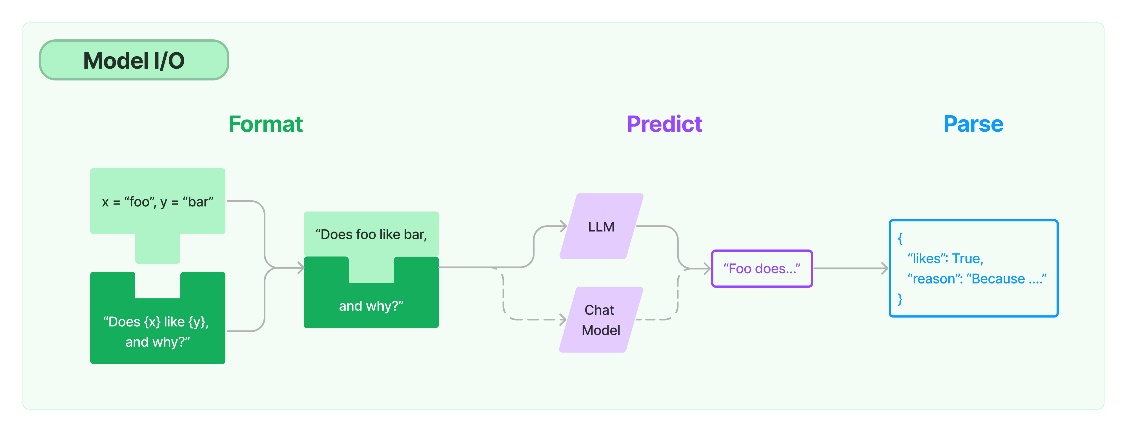


图 3-1

Model I/O 组件提供了三个核心功能（如图 3-1 所示）：

[语言模型](/docs/modules/model_io/models/): 通过接口调用语言模型，即 Model。

[提示](/docs/modules/model_io/prompts/): 将模型输入模板化、动态选择和管理，即 Model I。

[输出解析器](/docs/modules/model_io/output_parsers/): 从模型输出中提取信息 ，即 Model O。

我们将深入了解如何使用 Langchain 的模型输入输出组件来访问大语言模型。我们先认识三大模型类别。

语言模型的分类

LangChain 模型输入输出组件的目前有三大模型类型。分别是通用大型语言模型 LLMs， 专用聊天模型 Chat Models 和文本嵌入模型 Text Embedding Models。

在 LangChain 中使用的不同类型的完成大语言模型输入和输出。在这一章中，我们将对模型进行分类和认识。

通用大型语言模型

首先介绍的是大型语言模型（ LLMs ）。这些模型以文本字符串作为输入，并返回文本字符串作为输出。

专用聊天模型

第二种是专用聊天模型。这些模型通常由语言模型支持，但其 API 更加结构化。具体来说，这些模型以聊天消息列表作为输入，并返回聊天消息。

文本嵌入模型

文本嵌入模型是以文本作为输入，并返回浮点数列表，也就是向量维度表示的列表。

这三种类型中，通用大型语言模型和专用聊天模型很容易被人误解没有必要。实际上，通用大型语言模型和专用聊天模型有微妙但重要的区别。

通用大型语言模型和专用聊天模型的区别

LangChain 中的 通用大型语言模型是指纯文本补全模型， 也就是Text To Text。 它们包装的 API 接受字符串提示作为输入，并输出字符串补全部分。OpenAI 的 GPT-3 就是 LLM 的实现典型。

专用聊天模型通常由 LLMs 支持，但专门用于进行对话。

专用聊天模型的不同之处在于：提供以 “聊天消息” 作为输入和输出的接口。它们的输入不是单个字符串，而是聊天消息的列表。

通常，这些消息带有发言者身份（LangChain 目前支持的消息类型有“AIMessage”，“HumanMessage”，“SystemMessage”）。它们返回一个（“AI”）聊天消息作为输出。GPT-4 和 Anthropic 的 Claude 都是作为专用聊天模型实现的。

通用大型语言模型和专用聊天模型的的学习路径

在Langchain的发展迭代过程中，每个模块都精细地划分出通用大型语言模型（LLM）和专用聊天模型两种类型，紧跟OpenAI的前沿技术潮流，以更好地适配新的专用聊天模型。

这一区分已经形成了一种趋势，也为我们学习Langchain提供了线索。每当我们看到一个针对通用模型的Langchain类或方法，我们便能预期到对应的聊天模型类或方法也必然存在。

然而，Langchain这样的做法有时候会让人感到困惑，因为它每次都会单独声明通用大型语言模型和专用聊天模型的不同代码。但是，如果你从模型的输入输出开始理解，下面的学习过程会变得轻松很多。例如，如果你刚刚学到通用模型，你就预期会有一个聊天模型，你就能更好地做出预测。

模型输出的通用接口

无论是通用大模型还是专用聊天模型，Langchain 都暴露了predict方法，而且包装进了对话的链或者代理上，使用非常方便。

## 2.1.1 通用大型语言模型（LLMs）

大型语言模型（LLMs）是 LangChain 的核心组件。LangChain 不提供自己的 LLMs，而是提供了与许多不同的 LLMs 进行交互的标准接口。

我们开始逐个解释这些元素，通过编写代码来解析它们。

有许多 LLM 提供商（OpenAI、Cohere、HuggingFace 等）—— langchain.llms 类旨在为所有提供商提供标准接口。

使用 OpenAI LLM 包装器，这些方法对于所有 LLM 类型都是通用的。以下是示例代码：

安装和设置密钥

首先，我们需要安装 OpenAI Python 包：

pip install openai langchain

然后设置好密钥：

from langchain.llms import OpenAI  
OpenAI.openai\_api\_key = "YOUR\_OPENAI\_API\_TOKEN"

使用 LLM 的最简单方法: 字符串 in -> 字符串 out

# Run basic query with OpenAI wrapper  
llm = OpenAI()  
llm("Tell me a joke")

运行后的结果：

'Why did the chicken cross the road?\n\nTo get to the other side.'

说明： 这里的运行结果是随机的，而不是唯一固定的回答。

接下来我们转入第二种模型专用聊天模型。

## 2.1.2 专用聊天模型

当前最大的应用场景便是 “聊天 Chat” ，就像OpenAI的热门应用ChatGPT一样。为了紧跟用户需求，LangChain推出了专门专用聊天模型Chat Models，以便我们能与各种聊天模型进行无缝交互。

在上一节我们提到专用聊天模型的不同之处在于：提供以 “聊天消息” 作为输入和输出的接口。它们的输入不是单个字符串，而是聊天消息的列表。

这一节，我们通过代码，看看输入聊天消息列表的专用聊天模型, 究竟与通用 LLMs 有什么区别？

以下是示例代码：

安装和设置密钥

首先，我们需要安装 OpenAI Python 包：

pip install openai langchain

然后设置好密钥：

import os  
  
os.environ['OPENAI\_API\_KEY'] = ''

使用 Chat Models 的最简单方法

为了通过LangChain与聊天模型交互，我们将导入一个由三个部分组成的模式：一个AI消息 AIMessage，一个人类消息HumanMessage和一个系统消息SystemMessage。然后，我们将导入ChatOpenAI。

from langchain.schema import (  
 AIMessage,  
 HumanMessage,  
 SystemMessage  
)  
from langchain.chat\_models import ChatOpenAI

SystemMessage 是您在使用模型时用于配置系统的消息，而 HumanMessage 是用户消息。

我们现在将系统消息和人类消息组合成一个聊天信息列表，然后输入到 Chat Models 聊天模型。

这里我使用的是模型名称是：gpt-3.5-turbo。如果你有gpt4，也可以使用gpt4。

chat = ChatOpenAI(model\_name="gpt-3.5-turbo",temperature=0.3)  
messages = [  
 SystemMessage(content="你是个取名大师，你擅长为创业公司取名字。"),  
 HumanMessage(content="帮我给新公司取个名字，要包含AI的")  
]  
response=chat(messages)  
  
print(response.content,end='\n')

运行后获得 AIMessage 的结果是：

当然！以下是一些包含"AI"的创业公司名称的建议：  
  
1. AIgenius  
2. AItech  
3. AIvision  
4. AIpros  
5. AIlink  
6. AIsense  
7. AIsolutions  
8. AIwave  
9. AInova  
10. AIboost  
  
希望这些名称能够给你一些启发！如果你有其他要求或者想要更多的建议，请随时告诉我。

本节中，我们着重研究了如何运用专用聊天模型。在LangChain的持续迭代过程中，我们目睹了模型和LangChain都在不断进化。

## 2.1.3 文本嵌入模型

想要造航母，仅仅一个大语言模型是不够的。

假设我们想要创造一个《红楼梦》聊天机器人应用，并且询问关于这本书的知识。一个大语言模型就够了吗？

还有我们上传专业的论文，长达几十页的临床医学实验报告，或者使用搜索或外部数据的东西，跟真实的数据连接起来，大语言模型能处理吗？

虽然最新的模型一直在突破极限，但是仍然不能满足我们想要大模型阅读大型文档、书籍的需求。

面对这些挑战，我们可能需要总结或寻找其他方法来应对。因为大型语言模型可以传递的标记数量是有限的，大多数模型最多可以传递1024到16K个标记，尽管一些新的模型可以处理更多的标记。这就意味着，我们需要思考如何有效地利用这些有限的标记，以达到我们的应用需求。

此外，理解模型的类型和其各自的特点也同样重要。模型在高层次上可以分为两种不同类型：语言模型和文本嵌入模型。嵌入模型将文本转换为向量标记，让我们可以将文本标记为向量特征，显示在向量空间中。了解这两种模型的特性和差异，能帮助我们更好地选择和使用模型，进一步优化我们的应用。

嵌入模型将文本转换为向量标记的示例

在自然语言处理（NLP）中，嵌入模型是将文本转换为向量的重要工具，这样我们可以在向量空间中表示文本，以便进行后续的机器学习或深度学习任务。举个例子，我们可以用 Word2Vec，一个常见的嵌入模型，来演示这个过程。

假设我们有以下的语料库：

1. I love learning.  
2. I like reading books.  
3. Books are great for learning.

首先，我们需要建立一个词汇库，也就是列出所有在语料库中出现过的独立单词：

"I", "love", "learning", "like", "reading", "books", "are", "great", "for"

接下来，我们用 Word2Vec 来训练这个语料库。Word2Vec 是一个预测模型：它试图从上下文预测目标单词，或者从目标单词预测上下文。训练过程中，Word2Vec 会学习到一个单词和它的上下文单词之间的关系，并将这些信息编码到向量中。

训练完成后，每个单词都会被赋予一个向量。例如，“books”的向量可能是：

[0.1, -0.2, 0.3, 0.5, -0.1]

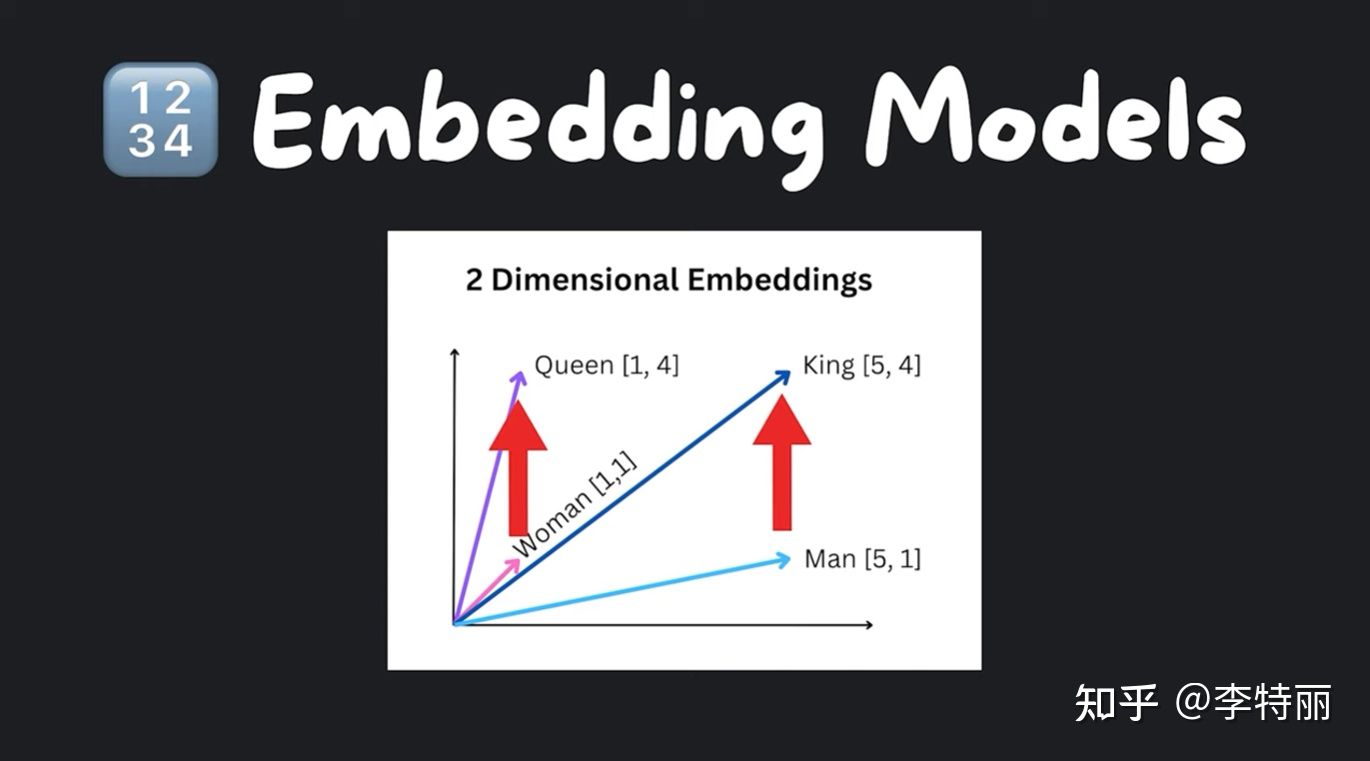
而“learning”的向量可能是：

[-0.1, 0.3, -0.2, 0.1, 0.4]

这些向量现在就代表了对应的单词在向量空间中的位置，且具有一些有趣的属性。比如，语义上相似的单词在向量空间中的位置会比较接近，我们可以通过计算两个向量之间的余弦相似度来衡量这种接近程度。

以上就是嵌入模型将文本转换为向量标记的一个基本例子。

嵌入模型的原理



在上面这个图像中，我们可以看到在一个二维空间中，“man”是“king”，“woman”是“queen”，它们代表不同的事物，但我们可以看到一种模式。这个模式就是可以在向量空间中寻找最相似的文本片段，实现语义搜索。

例如，OpenAI 的文本嵌入模型可以精确地嵌入大段文本，具体而言，8100 个标记，根据它们的词对标记比例 0.75，大约可以处理 6143 个单词。它输出 1536 维的向量。

我们可以使用 LangChain 与多个嵌入提供者进行接口交互，例如 OpenAI 和 Cohere 的 API，但我们也可以通过使用 Hugging Faces 的开源嵌入在本地运行，以达到 免费和数据隐私 的目的。

现在，您可以使用仅 4 行代码在自己的计算机上创建自己的模型嵌入。但是，维度数量可能会有所不同，嵌入的质量可能会较低，这可能会导致检索不太准确。

安装和设置密钥

我们需要安装 Langchain Python 包：

pip install openai langchain

使用 Text Embedding Models 的最简单方法

导入 OpenAIEmbeddings：

from langchain.embeddings import OpenAIEmbeddings  
  
embeddings = OpenAIEmbeddings(openai\_api\_key = "YOUR\_OPENAI\_API\_TOKEN")  
result = embeddings.embed\_query("This is an apple.")

打印 result ，我们就可以看到 “This is an apple.” 这句话的向量空间表达数组。

[0.012008527293801308,  
 -0.001835253438912332,  
 0.00145026296377182,  
 -0.0030227703973650932,  
 -0.005661344155669212,  
 0.005086636636406183,...]

打印出的数字并不能直观看到这些词在向量空间中的位置，但如果你对向量可视化感兴趣，可以访问OpenAI的官方网站，那里有许多关于词嵌入（embedding）可视化的实践场所（playground）供你探索和学习。