第 3 章

过程建模问题

3.1 问题描述

3.1.1 基准数据集

通过前两章的阅读，读者已经具备使用代码编辑器研究元学习算法的能力。从本章开始，章节内容的安排将在一定程度上进行升级——在问题描述阶段，即开始与代码注释结合。MAML 模型的原始代码主要采用了 2 个数据集，即 Mini-ImageNet 数据集和 Omniglot_resized 数据集。

Mini-ImageNet 数据集是 Google DeepMind 团队从 ImageNet 数据集中抽取的一小部分，大小约 3GB，共有 100 个类别，每个类别都有 600 张图像，共 60000 张（均是以.jpg 结尾的文件），而且图像的大小并不是固定的。

Omniglot_resized 数据集是 Omniglot 数据集的修正版，包含来自 50 个不同字母的 1623 个不同手写字符，构成 1623 个类别，其每个类别有 20 个样本，每个样本大小为 28×28 像素。原始代码采用的这 2 个数据集也是小样本学习常用的基准数据集。

制作 Mini-ImageNet 数据集，对普通研究人员或者开发者是一个友好的选择。ImageNet 数据集可以称得上深度学习革命的加速器，这是一个知名度非常高的开源、海量数据集。常见的目标检测、识别等算法，在完成设计后，通常需要在 ImageNet 数据集 1000 个类别的数据上进行训练以及验证。新模型的框架一般也需要先基于 ImageNet 数据集进行预训练，根据预训练模型做基线（baseline）评估。然而，这个数据集全部下载，大概有 100GB，训练过程对硬件要求也非常高，即使采用很多块高端显卡并行训练，也要花费好几天的时间。元学习模型进行了任务构建与联合训练，相比于深度学习，元训练过程对硬件要求要高出很多。因此，在元学习模型编程调试、算法应用改进中，建议采用 Mini-ImageNet 数据集代替 ImageNet 数据集。

要理解样本产生过程，就需要先了解数据集的结构。以 Mini-ImageNet 数据集为例，其根目录为 mini-imagenet，里面有 4 个子目录。第一个子目录是 images，所有的图像都存在该子目录中；其余 3 个子目录分别是 train.csv、val.csv、test.csv，分别用于保存训练集、验

证集、测试集的标签文件。文件格式.CSV 的英文全称为 Comma-separated Values，即逗号分隔值，主要用于在程序之间转换表格数据。这类文件以纯文本形式存储表格数据，也可以转换为 Excel 表格。

Mini-ImageNet 数据集每个 CSV 文件之间的图像以及类别相互独立，共 60000 张图像、100 个类别。作为元学习领域的基准数据集，标签文件并不是从每个类别中采样的。数据集的 64%用于训练、16%用于验证、20%用于测试。换言之，train.csv 子目录中包含 64 个类别的 38400 张图像，val.csv 子目录中包含 16 个类别的 9600 张图像，test.csv 子目录中包含 20 个类别的 12000 张图像。

MAML 模型的原始代码中提供了该数据集制作的模块代码，在 3.1.3 节会进行深入解读。理解数据集制作过程细节，再结合第 2 章的研究，可以完整地理解任务样本的产生过程。

3.1.2　图像尺寸调整

与 Mini-ImageNet 数据集的处理方式不同，Omniglot_resized 是对 Omniglot 中的图像尺寸调整后得到的数据集。图像尺寸调整也是服务于元学习过程建模的。Python 程序读取图像后，会将其转为矩阵向量的形式。在深度学习及元学习研究中，输入向量维数决定了输入层的网络节点数。因此，有必要统一输入向量的维度。通过一个常规的几何变换算法，就可以把 Omniglot 中的图像尺寸调整到相同的尺寸，以便于在元学习过程中进行标准化处理。在 MAML 模型的原始代码中，也提供了数据集 Omniglot 图像尺寸调整的模块，相关文件名为 resize_images.py，详细代码如下。

```
from PIL import Image
#从 PIL 库导入 Image 类,该类是 PIL 库中用于图像处理的函数
#在 Python 中,需要使用 PIL(Python Image Library)库处理图像

import glob
#导入 glob 模块,此模块可用于查找符合特定规则的路径名
#glob 是 global 的缩写,表示在 Windows 系统中进行全局搜索
#常用函数有 glob.glob()、glob.iglob()等,后者每次只能获取一个路径名

image_path = '*/*/'
#准备接收搜索到的图像路径,*用于打包位置参数
#*和**属于 glob 模块,是很灵活的符号,用于解包参数、扩展序列以及对字典和集合进行操作等
#不同的是,**侧重于解包关键字参数,并将这些参数打包成一个字典

all_images = glob.glob(image_path + '*')
#完成对所有图像的全局搜索,并将其位置参数打包成一个元组 all_images
#glob.glob()函数可以同时获取所有的匹配路径,并将这些路径返回至一个列表中

i = 0
```

```
#分类控制指标 i,其初始值为 0
for image_file in all_images:
#对图像路径列表中的所有文件做循环处理
#all_images 是图像路径列表

    im = Image.open(image_file)
#打开文件 image_file,并以矩阵向量形式赋值给 im,im 是 image 的简写
#Image.open()函数属于 PIL 库的 Image 模块,用于打开图像

    im = im.resize((28,28), resample = Image.LANCZOS)
#调用 im.resize()函数,以调整图像尺寸
#缩放过程中需要重采样
#im.resize()函数的用法是 im = im.resize(目标尺寸,重采样滤波器)
#resample,顾名思义,是重采样
#重采样过程中消除了锯齿噪声
#Image.LANCZOS 是重采样滤波器,可以抗锯齿噪声

    im.save(image_file)
#调用 im.save()函数,保存修改

    i += 1
#分类控制指标 i 的值加 1

    if i % 200 == 0:
#如果 i 除以 200 的余数等于 0,即 i = 200、400、600 等
#每 10 个类作为一个小样本单元,每个类有 20 个样本,故而每个单元有 200 个样本
        print(i)
#输出对应的 i 值
```

3.1.3 知识获取过程

所谓知识,其本质是一种元优化机制。这种知识的成功获取,具体体现在模型拥有了自动调整超参数的能力。在获取已有知识的基础上,模型可以快速学习新的任务。对于深度学习训练时间过长、参数微调难、新任务需要重复训练的问题,元学习模型设计的知识获取过程提供了一个全新的解决方案。知识获取的过程就是构建数据集的过程,即先生成一批服从某种分布模型的学习任务,然后通过对这批任务进行联合训练,可以得到一组较好的超参数,并赋予模型自主调参的先验知识。知识的自主应用过程对新的学习任务不用从头开始训练模型,只需要经过少数几次梯度下降,就可以完成超参数的自主微调,从而快速完成新的学习。元学习网络在现有神经网络结构的输入层、隐含层、输出层之外,新增了 meta 层,用于实现知识的获取过程。其获取的知识在整个学习过程中被全程应用,如图 3-1 所示。

图 3-1 中,θ 是元学习模型的参数,x 和 y 分别是模型的输入和输出,$\nabla_\theta L$ 表示任务样

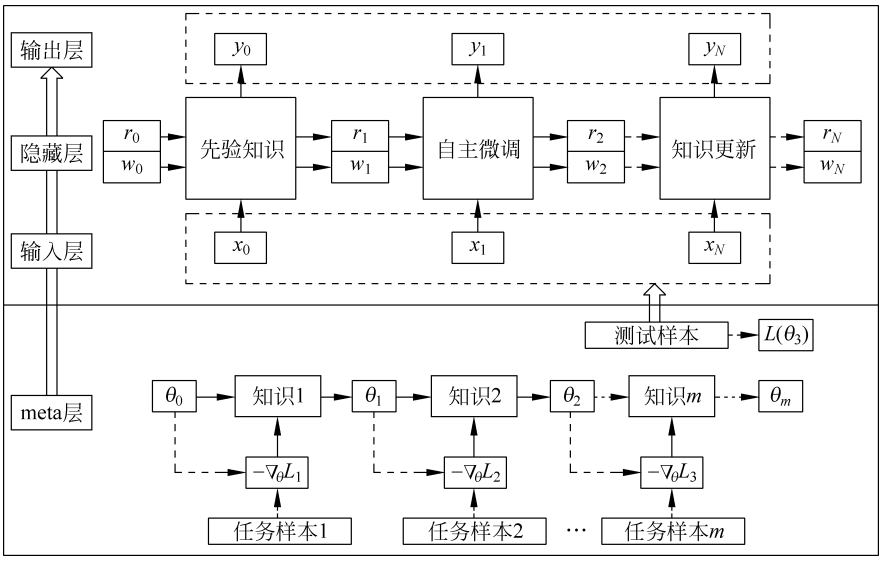

图 3-1　元学习模型的知识获取过程

本上的损失函数梯度，w、r 分别为模型权重系数及其对应的残差值(residual)。元学习的知识获取是模拟人类学会学习的过程，从而让机器学会学习。传统深度学习研究的模式是获取特定任务的海量数据集，并利用该数据集从头开始训练模型。这一训练过程构成深度神经网络的学习过程，与人类的学习过程相去甚远。人类在学习过程中，能够利用过去的经验，快速学习新任务。元学习的算法思想和训练过程主要借鉴类似的建模思路。对于每个不同的学习任务，不再需要重新训练模型。通过多次梯度下降，实现模型参数的自主微调，即可快速适应新任务。

3.2　建模思路

3.2.1　图像加载模型

知识获取始于对数据的学习，所以知识获取的第一步就是数据集加载与任务样本的输入。元学习模型的输入输出问题将在第 4 章进行研究，本节重点解释图像数据加载模型的建模思路。

图像加载模型需要考虑计算机的配置，因为图像尺寸决定进行运算处理时要求的配置。尺寸调整的基本目标是图像尺寸小于 500×500 像素，很多预训练模型甚至要求图像尺寸小于 300×300 像素。基于 3.1.2 节的代码注释，可按照如下步骤获取 Omniglot_resized 数据集。

1. 下载 Omniglot 数据集

首先，为 Conda 配置下载工具 wget，输入命令 pip install wget 即可(注意，pip 不可省略，否则会报错)。工具 wget 很小，只有 10kB，一般在两分钟内可以完成下载和安装，如

图 3-2 所示。

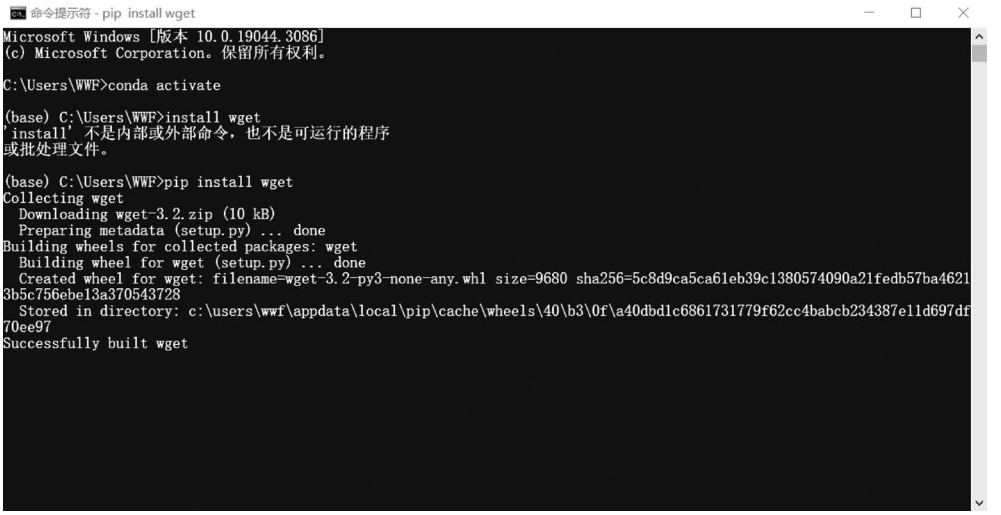

图 3-2 配置下载工具 wget

在 Python 编程中，wget 模块可以导入代码中，并用于下载数据集。顾名思义，wget＝web＋get，是通过网络下载。纽约大学助理教授 Brenden Lake 在 GitHub 网站上分享了专门用于 Python 的 Omniglot 数据集的下载链接。

本节的元学习模型代码调试主要使用其中的两个数据集——images_background 和 images_evaluation，Omniglot 数据集下载的前置代码如下所示。

```
import wget
url =  \
`https://github.com/brendenlake/omniglot/blob/master/python/images_background.zip'
#请注意，字符串符号的标记不能用引号，虽然 Word 中显示为引号

#指定一个保存路径，指定路径的下载代码为
path = "D:\mywgetdata"
#请注意，字符串符号的标记不能用引号，虽然 Word 中显示为引号
wget.download(url, path)
#调用 wget 模块的 wget.download()函数
#下载链接为 url，下载后保存路径为 path
```

参考第 2 章的方法，采用上述命令，编写代码文件 metadownload.ipynb，运行结果如图 3-3 所示。

选中一个代码行，可以看到该行结尾有 6 个按钮，分别用于复制、上移、下移、上方插入行、下方插入行、删除。Python 将这种代码行称为 cell，这些 cell 操作为编程提供的很多便利条件，有助于提高编程效率。此处，复制是 duplicate 操作，即插入一行完全相同的代码。如果需复制代码行，则需选中代码行，单击右上角“小剪刀”后的复制按钮即可。

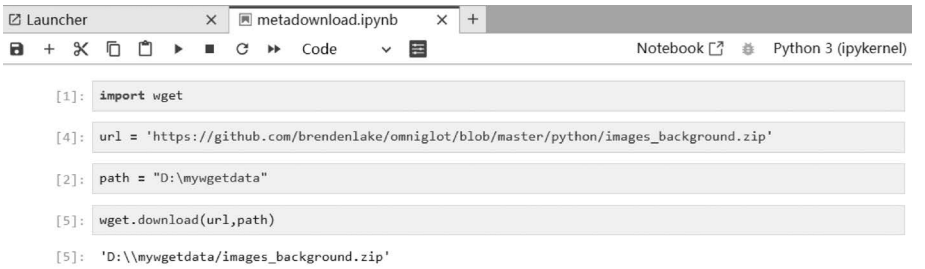

图 3-3　images_background.zip 的下载代码及运行结果

下载需要时间，请耐心等待，直到 mywgetdata 目录里出现 images_background.zip。不要重复单击运行按钮，否则可能出错，因为是网络下载，涉及网络协议。

将上述代码中的 url 链接替换为

```
url = https://github.com/brendenlake/omniglot/blob/master/python/images_evaluation.zip
```

如图 3-4 所示，重新运行代码文件 metadownload.ipynb，即可下载 images_evaluation.zip，下载完成后，将在目录中出现 images_evaluation.zip。当然，读者也可以直接单击链接，手动下载。

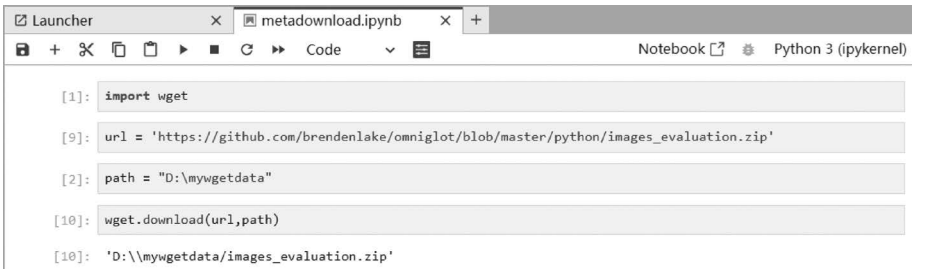

图 3-4　images_evaluation.zip 的下载代码及运行结果

如果希望同时下载 images_background.zip 和 images_evaluation.zip，则需要做一个 TXT 文档将两个链接都写进去。如果希望分别下载到不同的目录，也可以做一个 TXT 文档将两个路径写进去，感兴趣的读者可以去尝试。网络下载源是动态的，一段时间后，重新执行代码，可能会发现下载的两个压缩包均无法解压。此时尝试手动下载，会发现该下载链接的数据已经被上传人删除。幸运的是，还可以通过其他途径找到 Omniglot 数据集。

2. 对其子数据集进行解压

解压后，即可找到两个主要子目录 images_background、images_evaluation。感兴趣的读者也可以尝试用代码完成解压，Python 并没有提供 unzip()方法，但通过二次调用 zip()也可实现解压。

Step 3. 创建目录 data/omniglot，将 Step 2 得到的两个子目录复制到目录 data/omniglot 内。另一个数据集 Mini-ImageNet 的处理方法类似，下文不再赘述。

3.5.2 节会重点介绍如何配置 MAML 模型代码调试环境 Python＋Pycharm。图像尺

寸调整模型代码是已在 3.1.2 节注解的 resize_images.py，该代码不必执行。如感兴趣，可在命令行窗口执行，这些内容将在 3.3.1 节演示。

3.2.2 尺寸调整模型

3.1.2 节对图像尺寸调整代码 resize_images.py 进行了较为详细的注解，其核心调整代码为

```
im = im.resize((28,28), resample = Image.LANCZOS)
```

主要是通过调用 im.resize()函数，完成图像尺寸的调整。图像尺寸调整过程中，需要重采样(resample)。因此，im.resize()函数至少需要两个输入，第一个输入决定了经过调整后图像的尺寸，第二个输入决定了重采样时采用的滤波器。resize()函数还可以仅对感兴趣的区域(Region Of Interest，ROI)进行重采样，此时需要第三个输入，用法为 resize(size，resample，box)。要进行调整的图像区域参数 box，可以用 4 个坐标元组指定，核心调整代码改为

```
im = im.resize((300,200), resample = Image.LANCZOS, box = (0, 0, 150, 100))
#把(0,0)开始的 150×100 像素图像区域放大到 300×200 像素
```

resize()函数中的 size 表示图像的宽度和高度，单位为像素。这里采用的是 Image.LANCZOS 重采样滤波器，它可以较好地处理锯齿噪声。Lanczos 算法是匈牙利数学家 Cornelius Lanczos 在 20 世纪 40 年代建立的模型，这是一种用于计算矩阵特征值和特征向量的迭代算法，已经被广泛应用于科学和工程领域的大规模矩阵计算中。对于图像尺寸的调整问题，Lanczos 算法的主要建模思路是通过正交相似变换，将对称矩阵变成对称三角矩阵，从而在很大程度上降低重采样过程中的计算复杂性。每个图像的 Hessian 矩阵都是一个对称矩阵。除了 Lanczos 算法，重采样的常用算法还有双三次插值(bicubic interpolation)、双线性插值(bilinear interpolation)、最近邻插值(nearest interpolation)等。

调整图像尺寸之前，需要先打开图像，对应的核心代码为

```
im = Image.open(image_file)
```

用于打开图像的 Image.open()函数的用法为 open(filename，mode)。除了文件名 filename，另一个输入指定了打开的模式 mode，模式可以作为 im 的属性直接调用，调用格式为 im.mode。该模型还定义了 im 的像素类型、深度信息和色彩空间 RGB、HSV 等。并可以借助代码行 print(im，mode)运行输出。除了 size 和 mode，format 也是图像的重要属性，用于表示 JPEG、PNG 等图像格式。

3.2.3 空间插值模型

在深度学习模型发展的早期阶段，由于网络结构的限制，全连接层的输入维度是固定

的，所以必须把输入图像进行归一化变换，统一为固定的尺寸。得到模型输出后，还可以再借助反变换，还原到原来的尺寸。这种早期的修正方法主要是为了适应卷积神经网络，但是随着对模型精度的进一步研究，科学家开始意识到归一化修正所带来的损失，空间插值模型由此应运而生。

事实上，只需要确保任意维度的图像都可以转换成固定的维度，就能够适应全连接层的输入，从而解除网络结构对输入图像尺寸的依赖。此外，还有一个建模思路是舍弃全连接层，直接构建全卷积神经网络。此时，输入图像的尺寸不受限制，因为卷积操作是一种类似滑动窗口扫描的处理方式。在元学习模型代码中引入调整问题，主要是考虑训练阶段的问题。模型假设训练集和测试集的大小一致，而且 batch 的读取方式也要求保持训练图像尺寸一致。

im.resize()函数可以应用的空间插值模型有很多种，除了 3.2.2 节中提到的最近邻插补模型、双线性插值模型、双三次插值模型和 Lanczos 插值模型，还有面积插值模型 AREA，即根据像素面积相关性完成重采样。空间插值模型的运用从本质上改变了图像的特征细节，在元学习模型研究过程中要注意灵活运用。当元学习模型精度达不到预期效果时，可以考虑选择不同的空间插值模型，特别是元学习面临未知的新任务及其相关的陌生图像，空间插值模型的选择将直接决定训练结果的好坏。对于质量较低的大规模数据集，空间插值模型导致的差异可能会非常明显。然而，不能简单地通过少数几次实验论证空间插值模型的优劣。一般可以借助公开的数据集做初步的评估，然后结合实际应用效果，为对应的项目选择适合的空间插值模型。

3.3　算法思想

3.3.1　文件保存算法

在 3.1.2 节注解的代码中，尺寸调整的结果需要调用 im.save()函数，以完成图像文件的保存，即

```
im.save(image_file)
```

本节主要关注代码实际应用过程中的文件保存算法。尝试在 3.1.2 节中注解代码文件 resize_images.py 的原始用法，如果遇到困难，进一步理解并修改即可。参考该代码公开的原始用法，文件处理和保存算法的完整实现步骤如下。

Step 1. 打开命令行窗口，输入命令“d:”，切换到保存数据集的磁盘空间，如图 3-5 所示。注意确认保存数据集的硬盘名称。

图 3-5　文件保存算法——Step 1-从命令行窗口切换到保存数据集的磁盘空间

Step 2. 从命令行窗口输入保存数据集的完整路径，即可进入保存数据集的目录。

```
cd D:\Metalearning－从最优化到元优化\code\第 3 章\data\omniglot
＃cd 命令适用于 DOS 系统、Linux 系统及 Windows 系统，用于进入指定的目录
```

注意确认保存数据集的路径修改，否则会有错误提示——系统找不到指定的路径。运行成功后，会显示当前已进入的目录，如图 3-6 所示。

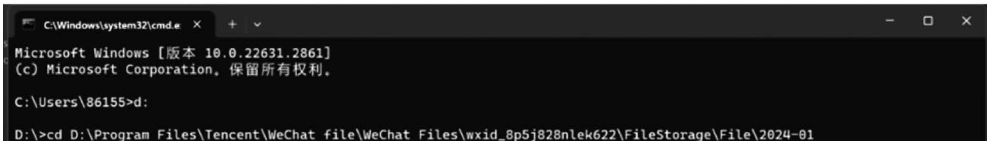

图 3-6　文件保存算法——Step 2-从命令行窗口进入保存数据集的目录

现在可以运行此目录中的代码了。确认目录 omniglot 是否已经按照 3.2.1 节的要求进行处理，即是否已经包含了 images_background 和 images_evaluation 两个子目录。确认后，复制尺寸调整算法代码文件，即 3.1.2 节注解的 resize_images.py，并备份到目录 data/omniglot 内。

Step 3. 在命令行窗口，输入命令 python resize_images.py，按 Enter 键执行。运行结果如图 3-7 所示，该代码涉及尺寸调整。

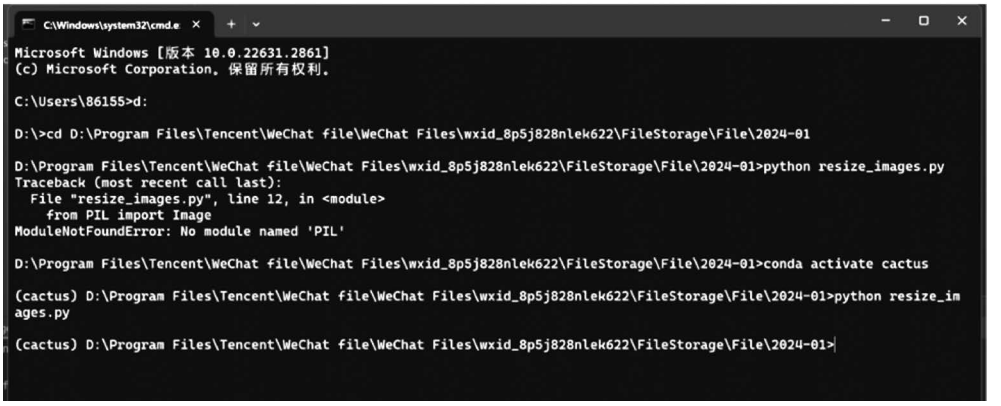

图 3-7　文件保存算法——Step 3-resize_images.py 运行成功

元学习代码调试中，无须提前单独运行 resize_images.py 和 proc_images.py 代码文件对数据做预处理，因此，读者不必尝试运行此代码。此外，必须在执行命令之前检查并确认数据集目录及其子目录里的文件均为非只读状态，否则会被拒绝修改，进而导致运行失败。

3.3.2　目录创建算法

为方便在元学习过程中调用数据，完成样本任务的构建与联合训练，需要进行文件目录创建。在 Python 编程中，可以导入 os 模块达成这些目标。模块 os 称为标准模块，可用于实现以下目录的创建。

(1) os.getcwd()函数用于获取当前工作目录；

(2) os.mkdir()函数用于创建新的目录；

(3) os.makedirs()函数用于递归创建目录；

(4) os.remove()函数用于删除指定文件；

(5) os.rmdir()函数用于删除非空目录；

(6) os.listdir( )函数用于返回指定目录下文件或子目录名字的列表。

特别地，MAML 模型代码对于 Mini-ImageNet 数据集的处理涉及目录创建，现仍然采用模块切割的办法，将代码分解为 5 个简单的小模块，以便做通俗易懂的解释。其对应的核心代码及其算法思想详细注解如下。

```
"""
        #第一个模块导入目录创建过程中所需的科学计算包
"""

from __future__ import print_function
#从 __future__ 模块导入 print_function 输出特性
#该特性将 print 语句解析为 print()函数,以便在当前安装的版本中使用最新版的 print()函数
#模块_future_很灵活,且很强大
#该模块的原理是,通过修改 Python 解释器的行为以适应未来的 Python 版本。相较于当前 Python
#版本,未来版本可能增加或修改一些特性
#模块_future_提供的新特性带有预测性质,但被广泛认可。甚至有观点认为,_future_模块中存在
#的特性终将成为 Python 语言标准的一部分,届时将不再需要使用 Python 的_future_模块

import csv
#导入 Python 自带的 csv 模块
#该模块的 open()函数可打开 CSV 文件,并与 csv.reader()、csv.writer()函数结合,读取和写入文件
#目录信息源于 CSV 文件,此文件格式的英文全称为 Comma - Separated Values,即逗号分隔值,主要
#用于在程序之间转移表格数据。这类文件以纯文本形式存储表格数据,也可转换为 Excel 表

import glob
#导入 glob 模块。该模块可用于查找符合特定规则的文件路径名,glob 是 global 的缩写,即在
#Windows 下进行全局搜索。常用函数有 glob.glob()、glob.iglob()等,后者每次只能获取一个路
#径名

import os
#导入 os 模块,以便完成对文件或目录的操作

from PIL import Image
#从 PIL 库导入 Image 类。Image 类用于图像处理
#在 Python 中,需要使用 PIL 库来处理图像。PIL 英文全称为 Python image library

"""
        #第二个模块获取目录创建过程中所需的文件路径名
```

```
"""

path_to_images = 'images/'
#准备接收搜索到的图像路径

all_images = glob.glob(path_to_images + '*')
#完成所有图像的全局搜索,并将其路径位置参数打包成一个元组 all_images
#glob.glob()函数可以同时获取所有的匹配路径,并将这些路径返回至一个列表中
# * 用于打包位置参数, + 用于连接字符串,避免路径名的混连
# * 和 ** 属于 glob 模块,是很灵活的符号,用于解包参数、扩展序列以及对字典和集合进行操作等
#不同的是, ** 侧重于关键字参数的解包,并将这些参数打包成字典
```

3.3.3 文件读取算法

目录创建过程中,应同时进行文件的读取,以确保算法效率。

如前所述,尺寸调整是数据集处理的必要环节。在深度学习及元学习研究中,输入向量维数决定了输入层的网络节点数。3.1 节和 3.2 节解释了 Omniglot 数据集的尺寸调整方法,Mini-ImageNet 数据集的处理方法也是类似的。不同的是,Mini-ImageNet 数据集还涉及目录创建和文件读取的过程,其目录创建算法已在 3.3.1 节进行了解释。

在文件读取之前,需要先执行图像尺寸调整,对应的核心代码及其注释如下。

```
"""
#第三个模块完成文件读取之前所需的图像尺寸调整操作
"""
for i, image_file in enumerate(all_images):
#对图像路径列表中的所有文件做循环处理
#enumerate()函数用于枚举
#单词 enumerate 有枚举、列举的含义
#与 Omniglot 的图像尺寸调整方法不同,此处的 i 与 image_file 同步进入 for 循环
#all_images 是图像路径列表

im = Image.open(image_file)
#打开文件 image_file,并以矩阵向量形式赋值给 im,im 是 image 的简写
#Image.open()函数属于 PIL 库的 Image 模块,用于打开图像

im = im.resize((84, 84), resample = Image.LANCZOS)
#调用 im.resize()函数,调整图像尺寸,此处该函数的目标尺寸是(84,84)
#resample,顾名思义,是重采样,重采样过程中消除了锯齿噪声
#Image.LANCZOS 是重采样滤波器,可以抗锯齿噪声
#缩放过程中需要借助重采样滤波器,以便保持图像质量

im.save(image_file)
```

```
    #调用 im.save()函数,以保存修改

    if i % 500 == 0:
    #如果 i 除以 500 的余数等于 0,即 i = 500、1000、1500 等
    #每 100 个类作为一个小样本单元,每个类有 5 个样本,故而每个单元有 500 个样本

    print(i)
    #输出对应的 i 值

    """
    #第四个模块完成目录创建过程中的文件读取
    """
    for datatype in ['train', 'val', 'test']:
    #3 轮循环,依次是训练集、验证集和测试集
    #datatype 有数据类型的含义,此处将数据归并为 3 类

    os.system('mkdir ' + datatype)
    #在当前系统下,为当前循环的 datatype 创建目录
    #os.system()函数主要用于执行与操作系统相关的命令
    #os.mkdir()函数用于创建新的目录
    #对应当前循环的 datatype 创建
    #循环结束后将产生 train、val、test 3 个目录
    #此处, + 用于连接字符串,避免目录名称的混连

    with open(datatype + '.csv', 'r') as f:
    #在为当前循环的 datatype 打开对应的 CSV 文件,完成冒号后的操作之后,自动关闭文件
    #在下述代码中,被打开的 CSV 文件将被简写为 f
    #with open()函数用于打开文件,并在使用完后自动关闭,以免后续编辑权限受到限制
    #此处, + 用于连接字符串,避免 datatype 名称的混连

    reader = csv.reader(f, delimiter = ',')
    #调用 csv.reader()函数,完成 with open()函数对应的操作
    #操作对象为 f,即上一行代码中打开的 CSV 文件
    #该操作类型在调用 with open()函数时已经声明
    #上一行代码中的 r 是 reader 的简写
    # 单词 delimiter 有分隔符的含义
    #在文件读取过程中采用逗号分隔符

    """
    #第五个模块完成文件读取过程中的标签分配,该模块是第四个模块的内嵌模块
    """
        last_label = ''
    #定义标签分配的收尾符
    #标签分配结束后,以''结尾
            for i, row in enumerate(reader):
```

```
#对 reader 列表中的所有字符串做循环处理
#enumerate()函数用于枚举
#单词 enumerate 有枚举、列举的含义
#reader 列表中包含了 i 与 row,这两个指标同步进入 for 循环
#reader 列表已在文件读取的外层循环中定义
            if i == 0:
#冒号前为操作前提
                Continue
#冒号后为具体操作
#如果 i == 0,则跳过本次循环,返回至第一行

            label = row[1]
#第一行是标签字符串

            image_name = row[0]
#标签名称的前一行,即第 0 行,为图像名称

            if label != last_label:
#如果当前标签不等于收尾符,则执行冒号后的操作
                cur_dir = datatype + '/' + label + '/'
#具体操作是标签分配,首先是 datatype 与 label 的匹配,并打包为 cur_dir
#此处,+用于连接字符串,避免 datatype 名称与 label 字符串的混连
                os.system('mkdir ' + cur_dir)
#继续完成操作,现在将目录字符串归并到 cur_dir
#至此,为已创建的目录分配 datatype、label 已完成,下一步还需要对应到图像文件

                last_label = label
#标签分配完成

            os.system('mv images/' + image_name + '' + cur_dir)
#将 image_name 字符串归并到 cur_dir
#至此,完成目录创建的最后一步
#此处, +用于连接字符串,避免在归并过程中导致字符串的混连
#mv 是 os.system()函数中的命令,可以实现很灵活的操作
# 用法为 os.system('mv 文件名称' + 路径名称)
```

3.4 最优化方法

3.4.1 随机抽样过程

元学习过程建模的关键思想是,通过随机抽取元训练数据中的类,划分小样本单元,完成任务构建,以进行联合训练,最终提取跨任务的元知识。因此,随机抽样方法的选择是过程建模的关键一步。在开源的 MAML 模型代码中,主要通过 Python 中的 random 模块实

现随机抽样，并在随机抽样的基础上构建一批小样本学习任务，然后借助样本学习、最优化的方式完成每一个学习任务。

现将过程建模中的随机抽样环节、样本学习环节的代码及其对应的最优化方法依次详细注解，并将代码分解为 6 个简单的小模块，以便读者轻松理解元学习过程建模的细节，如下所示。

```
"""
        #第一个模块导入元学习过程建模所需的科学计算包
"""

import numpy as np
#将 NumPy 库导入为 np
#import as 句式在导入科学计算包的同时，提供了简写
#后续代码行可以用 np 表示 numpy
#该工具包是 Python 开源的高级科学计算包，可用于元学习模型编程
#能对数组结构数据进行运算，实现对随机数、线性代数、傅里叶变换等的操作
#它不只是一个函数库，更是拥有强大的 n 维数组对象——numpy 数组，也称为 ndarray 数组
#ndarray 数组由两部分构成——真实数据和描述真实数据的元数据，可通过索引或切片来访问和修改

import os
#导入 os 模块，以便对目录中的图像进行抽样

import random
#导入 random 模块，该模块主要用于生成随机数
#任务构建过程中需要从数据集分布 D 中随机采样

import tensorflow as tf
#导入 tensorflow 模块，并简写为 tf
#TensorFlow 是一个端到端的开源机器学习框架

from tensorflow.contrib.layers.python import layers as tf_layers
#从 tensorflow.contrib.layers.python 模块导入 layers 工具，并简写为 tf_layers
#layers 工具主要用于访问模型的层次细节，从而在元学习过程中可以轻松完成建模

from tensorflow.python.platform import flags
# 从 tensorflow.python.platform 模块导入 flags 工具
#flags 工具主要用于定义、获取命令行参数，以便完成样本学习

FLAGS = flags.FLAGS
#引入全局参数 tensorflow.python.platform.flags.FLAGS，并简写为 FLAGS
#在样本学习环节，将使用 FLAGS 解析命令行参数


"""
```

```
        #第二个模块实现元学习过程中的随机抽样环节
"""

def get_images(paths, labels, nb_samples = None, shuffle = True):
#定义抽样函数 get_images()
#get_images()函数有 4 个输入,前两个输入分别是路径、标签
#第三个输入是要抽取的样本数,nb_samples 是 number of samples 的简写
#None 表示 nb_samples 被解析为空值,等待输入
#第四个输入是洗牌方法,即打乱次序的方式
#True 表示每次都会返回不同的次序
#get_images()函数将采用随机抽样算法

if nb_samples is not None:
#当输入的 nb_samples 值满足条件时,执行此冒号后的操作
  sampler = lambda x: random.sample(x, nb_samples)
#从路径 x 无放回随机抽样,抽取 nb_samples 个样本,保存到小样本单元 sampler。基于小样本单
#元的任务构建是通过循环调用此代码模块而完成的
#random.sample()函数用于无放回地随机抽样
#函数 lambda 是 Python 的一个很灵活的创新,被称为匿名函数,有时候也称为 lambdas
#用法为 lambda 输入变量:函数表达式
#在任意一行代码中,函数 lambda 允许随时进行嵌入式定义,只需要一个表达式就可以完成表达式
#的计算结果也很方便得到,直接将输入变量值代入表达式即可

else:
#当输入的 nb_samples 值没有满足条件时,执行此冒号后的操作
sampler = lambda x: x
#将 None 保存到小样本单元 sampler,此时 x = nb_samples = None
#  lambda 后变量的含义由对应的表达式决定,此时的 x 不表示路径

images = [(i, os.path.join(path, image)) \
#对每个文件 image,进行 os.path.join(path, image)操作,并将结果归并到 images
#此番归并,将配合下一步的 for 循环进行迭代,以完成每个小样本单元的标签分配
#os.path.join()函数是路径操作函数,可以将多个路径拼接、合并为一个新的路径
#此处,os.path.join()函数实现 path 与 image 路径的拼接
#依据下一步 for 循环代码,可以将此处的 path 理解为标签 labels 的路径
#与下一步的循环控制指标 i 配对,标签和 image 路径合二为一,从而预设了标签分配格式

  for i, path in zip(labels, paths) \
#zip(labels, paths)中的每一对指标 i 和 path 进入同步循环
#zip()函数函数能将多个可迭代对象打包成一个元组,而每个元组包含来自所有可迭代对象的相
#同索引位置上的元素。所以 zip(classes, labels)将标签和类别的字符串索引合二为一,在索引上
#达成了预设的标签分配格式

  for image in sampler(os.listdir(path))]
#将小样本单元 sampler 中的每一个 image 加入嵌套循环
```

```
    #os.listdir(path) 返回 path 目录下所有 image 的列表

    if shuffle:
    #如果条件满足,执行此冒号后的操作
    #笔者认为,if shuffle 在这里的含义是 if shuffle == True
      random.shuffle(images)
    #将 images 次序打乱,然后随机抽样

       return images
    #至此,已完成 get_images()函数的定义,该函数的返回值为抽取的 images
```

3.4.2　样本学习过程

如前所述,过程建模主要包括随机抽样、样本学习及其对应的最优化过程。在 3.4.1 节中,已经详细注解了随机抽样的代码。样本学习主要是通过卷积操作和最大池化操作来实现,在元学习系统完成卷积后,需要先经过归一化过程,之后才进行最大池化操作。

在解释样本学习代码模块之前,需要先介绍一下步长(stride)的概念。在 Python 中,stride 作为一个有趣的概念被引入,并用于提高图像处理算法的性能,主要是基于两方面的原因。一方面,图像矩阵数据排列紧密,如按行操作会频繁读取非对齐内存,从而影响效率。另一方面,CPU 的工作原理也要求内存访问对齐,否则会触发硬件非对齐访问错误。

当 CPU 需要取 m 个连续字节时,若内存起始位置的地址可以被 m 整除,则称为对齐访问。若不被整除,则称为非对齐访问。由于图像维度的多样性,非对齐访问几乎是无法避免的。stride 的概念限定了图像矩阵中一行元素所占存储空间的长度,实现了强制性的对齐访问。|stride|≥图像宽度(byte)值,有可能会造成部分内存浪费。因此,使用时要确保内存充裕。

在样本学习环节,stride 是一个有符号数,可以理解为卷积操作和池化操作的步幅。如从下而上滑动,图像将拥有一个负的 stride 值。stride=m,相当于把图像尺寸缩小到原来的 $1/m$,处理速度提升了 m 倍。代码里的 stride 是长度为 4 的一维向量,即 stride=[batch, horizontal, vertical, channel]。步长的 4 个分量依次代表 batch、水平、垂直、channel 4 个维度上的滑动步长。其中,batch 维度是小样本单元维度,当 batch=1 时,不会跳过任何一个样本;channel 维度是颜色通道维度,当 channel=1 时,不会跳过任何一个颜色通道。

Tensorflow 框架中一般采用 stride = [1,1,1,1] 或 stride = [1,2,2,1]。在元学习过程建模代码中,两者皆被采用。但是,令 stride =[1,2,2,1],将 [1,1,1,1] 记为 no_stride,可突出 2 种步幅的差异。当最大池化方式为 VALID 时,采用 [1,2,2,1],否则采用[1,1,1,1]。事实上,4 个维度的分量均为 1,即没有任何跳过的情况,就是 no stride。由此可见,stride 主要用于协助完成样本学习过程中的最大池化操作。在 TensorFlow 框架下,最大池化操作可以借助 tf.nn.max_pool()函数实现。

tf.nn.max_pool()函数的具体用法为 tf.nn.max_pool(conv_outp，stride1，stride2，max_pool_pad，name)。此处，第一个输入 conv_outp，是卷积层的输出结果，一般是先卷积再池化。stride1、stride2 分别是池化、卷积操作的步幅，max_pool_pad 是最大池化的方式。第五个输入是操作的名称 name。其中，pad 是 padding 的简写，有 SAME 和 VALID 两种方式。受限于池化窗口大小和步幅，图像部分区块可能缺失。SAME 方式是对池化后的图像矩阵进行补零操作，而 VALID 方式是进行舍弃操作，2 种方式得到的输出维度不同。此函数的返回值是特征图 feature map。在 TensorFlow 框架下被解释为一个四维张量 Tensor=[batch，height，width，channels]。

现在，将解释过程建模代码的第三个模块。

```
"""
        #第三个模块实现过程建模中的样本学习过程
"""

def conv_block(inp, cweight, bweight, reuse, scope, activation = tf.nn.relu, max_pool_pad =
'VALID', residual = False):
#定义卷积模块函数 conv_block(),conv 是 convolutional 的简写,单词 convolutional 有卷积的含义
'''
#conv_block()函数有 8 个输入,分别为 inp、cweight、bweight、reuse、scope、activation、max_pool_
pad 和 residual。其中,inp 是输入样本 input 的简写,cweight 是卷积层的权重系数,bweight 是与
偏差 bias 对应的权重系数,reuse 规定了这些参数是否重复使用,scope 给出了这些参数共享的范
围,activation、max_pool_pad、residual 分别代表激活函数、最大池化空间、是否考虑残差
'''
#pad 是 padding 的简写,单词 padding 有填充的含义,这里定义了学习内容之间的空间属性

stride, no_stride = [1,2,2,1], [1,1,1,1]
# 定义两种滑动窗口:有跳跃 stride = [1,2,2,1],无跳跃 no_stride = [1,1,1,1]
#引入 no_stride,以减少 stride 造成的内存浪费

    if FLAGS.max_pool:
#当条件满足时,执行此冒号后的操作
# 条件 if FLAGS.max_pool 是 if max_pool_pad == 'VALID'的简写
#在第二个模块已经引入全局参数 tensorflow.python.platform.flags.FLAGS,简写为 FLAGS
conv_output = tf.nn.conv2d(inp, cweight, no_stride, 'SAME') + bweight
#执行的操作为卷积,输出结果为 tf.nn.conv2d(inp, cweight, no_stride, 'SAME') + bweight
#在最大池化方式为 VALID 的前提下,无须跳跃,直接调用 tf.nn.conv2d()函数,计算 2D 卷积结果
#因为 no_stride = [1,1,1,1],此时最大池化方式为 VALID,所以卷积过程中无须跳跃
#tf.nn 是 TensorFlow 专为神经网络设计的模块化接口,具体包含卷积、池化、线
#性等操作和损失函数 loss 的计算等,nn 是 neural network 的简写
# tf.nn 中的每一个模块都是神经网络层次化结构中的某一层。conv2d 是其中的二维卷积层
#'SAME'表示采用 SAME 卷积计算公式,即在卷积输出时,特征图的尺寸保持不变
```

```
else:
#当条件没有满足时,执行此冒号后的操作
conv_output = tf.nn.conv2d(inp, cweight, stride, 'SAME') + bweight
#在最大池化方式为 SAME 的前提下,调用 tf.nn.conv2d()函数,先跳跃,再计算 2D 卷积结果
#因为 stride = [1,2,2,1],此时最大池化方式为 SAME,所以卷积过程中需要分别在水平维度和垂
#直维度进行跳跃,跳跃的步幅为 2

#以上代码是归一化之前的卷积操作
 normed = normalize(conv_output, activation, reuse, scope)
#归一化过程可调用 normalize()函数实现,该函数是自定义函数
#normalize()函数有 4 个输入,分别为 conv_output、activation、reuse 和 scope,是自定义函数,其
#具体定义将在过程建模代码的第四个模块进行展示
#归一化之前的卷积结果 conv_output 也作为 normalize()函数的输入

#以下代码为归一化之后的最大池化操作
if FLAGS.max_pool:
#当条件满足时,执行此冒号后的操作
# 条件 if FLAGS.max_pool 是 if max_pool_pad == 'VALID'的简写
  normed = tf.nn.max_pool(normed, stride1, stride2, max_pool_pad)
#池化过程可调用函数 tf.nn.max_pool 实现,这是一个自定义函数
# tf.nn.max_pool()函数有 4 个输入,分别为 normed、stride1、stride2 和 max_pool_pad
#最大池化结果仍然记为 normed,因为最大池化是在归一化之后进行的
#池化和卷积均采用同一个步幅 stride = [1,2,2,1]
#此时,最大池化方式为 VALID,输出图像矩阵中缺失的元素将被舍弃

return normed
#卷积模块函数 conv_block()的定义到此结束
#该函数的返回值是归一化之后的最大池化结果

"""
        #第四个模块定义样本学习环节的归一化过程
"""

def normalize(inp, activation, reuse, scope):
#normalize()函数有 4 个输入,分别为 inp、activation、reuse 和 scope。其中,inp 是输入样本
#input 的简写,activation 代表激活函数,reuse 规定模型参数是否重复使用,scope 给出这些参数
#共享的范围

  if FLAGS.norm == 'batch_norm':
  #当此条件 1 满足时,执行此冒号后的操作
#这里 batch_norm 用于计算模型的 BN 层(英文 batch normalization,批归一化处理层)
#BN 层与激活函数层分别是卷积与池化的前置层
  return tf_layers.batch_norm(inp, activation_fn = activation, reuse = reuse, scope = scope)
#此时,normalize()函数的返回值为 tf_layers.batch_norm()函数的当前计算结果
```

```
    #tf_layers.batch_norm()函数用于计算 BN 层,有 4 个输入,分别为 inp、activation_fn、reuse 和
    #scope。此处,inp 是 input 的简写,activation_fn = activation 指定当前激活函数为默认的
    #activation,设定 reuse = reuse 和 scope = scope 是为跟踪参数。其中,reuse 规定模型参数是否
    #重复使用,scope 给出参数共享的范围。BN 层用于反馈中间层数据分布的变化

    elif FLAGS.norm == 'layer_norm':
    #当此条件 2 满足时,执行此冒号后的操作
    #此处的 layer-norm 也是用于计算模型的 BN 层,有两种归一化维度不同的计算方式,都是先计算
    #一个 batch 中所有通道的参数均值和方差,然后进行归一化。在自然语言处理中只需要使用输入
    #张量的第一维度,而在图像处理中对维度的操作则采完全不同
    #张量有一维、二维、三维、四维等。图像矩阵就是二维张量,加入深度信息就成为三维张量,一批三
    #维张量可以打包成一个四维张量
    return tf_layers.layer_norm(inp, activation_fn = activation, reuse = reuse, scope = scope)
    #此时,normalize()函数的返回值为 tf_layers.layer_norm()函数的当前计算结果
    '''
    #tf_layers.layer_norm()函数用于计算 BN 层,有 4 个输入,分别为 inp、activation_fn、reuse 和
    #scope。此处,inp 是 input 的简写,activation_fn = activation 指定当前激活函数为默认的
    #activation,设定 reuse = reuse 和 scope = scope 是为跟踪参数。其中,reuse 规定模型参数是否
    #重复使用,scope 给出参数共享的范围。BN 层用于反馈中间层数据分布的变化
    '''
    #如果条件 1 和条件 2 都没有满足,必然满足条件 3
    elif FLAGS.norm == 'None':
    #条件 3 FLAGS.norm == 'None'表示未明确如何计算模型的 BN 层
    #当条件 3 满足时,执行此冒号后的操作

    #分两种情况:
      if activation is not None:
    #条件 3 - 1 表示未明确如何计算模型的 BN 层,但是指定了激活函数
    #当不满足条件 1 和条件 2,但满足条件 3 - 1 时,执行此冒号后的操作
    return activation(inp)
    #此时 normalize()函数的返回值为激活函数的当前计算结果

      else:
    #此时,不仅未明确如何计算模型的 BN 层,也没有指定激活函数
    #当不满足条件 1 和条件 2,且不满足条件 3 - 1 时,执行此冒号后的操作
    return inp
    #此时 normalize()函数的返回值为当前的输入结果
```

3.4.3 最优化过程

本书以问题为导向,对元学习模型的建模思路、算法思想、最优化方法和元优化机制展开系统化的研究。以元学习的经典模型 MAML 为例,前两章主要探讨元学习模型中的联合训练问题与任务构建问题,第 3 章主要分析过程建模问题,最后两章将分别研究元学习模型的输入输出问题和应用拓展问题。通过前面章节的阅读,读者已经获得必要的前期积累。

接下来，将以通俗易懂的语言，并结合对应的模型代码，解释元学习模型的最优化方法，即损失函数作为最优化目标的收敛方法。

```
"""
        #第五个模块定义过程建模中的第一类最优化目标——均方误差 MSE
"""

def mse(pred, label):
#定义 mse()函数，顾名思义，是将损失函数定义为均方误差
#均方误差 MSE 的英文全称为 mean - square error，用于反映预测值与真实值之间的差异程度
#在第 2 章对任务构建问题的分析过程中，已将关注焦点集中在监督学习
#预测值 pred 是模型输出的标签，真实值 label 是图像的真实标签
#pred 是 predicted label 的简写，单词 predicted 有预测的含义

pred = tf.reshape(pred, [-1])
#无法预知 pred 当前的 shape
#但目标是明确的，希望重新调整为一列

label = tf.reshape(label, [-1])
#无须关注 label 当前的 shape
#目标是明确的，希望重新调整为一列

  return tf.reduce_mean(tf.square(pred - label))
#mse()函数的返回值为 tf.reduce_mean(tf.square())函数的当前计算结果
#  tf.square(pred - label)是调用 square 模块计算 pred - label 中每一个元素的平方值
#tf.reduce_mean()函数用于计算张量 tf.square(pred - label)沿指定数轴上的平均值，指定的数
#轴是指该张量的第一维度。tf.reduce_mean(tf.square(pred - label))函数构成了均方误差 MSE
#的计算公式


"""
        #第六个模块实现过程建模中的第二类最优化目标：交叉熵 XENT
"""

def xent(pred, label):
#定义 xent()函数，顾名思义，是将损失函数定义为交叉熵
#交叉熵 XENT 的英文全称为 cross entropy，用于反映预测值与真实值概率分布间的差异程度
#交叉熵损失函数 xent()作为最优化的目标函数，具有明显优势
#可以根据交叉熵损失函数 xent()的二阶导数，判断最优化目标函数的凸性或者凹性
#根据最优化目标函数的凸性或者凹性，可判断元学习模型训练过程中是否会陷入局部最优解
```

```
    #注意 TensorFlow 的版本,低版本的 TensorFlow 可能计算出错误的二阶导数。3.5 节将给出解决方案

        return tf.nn.softmax_cross_entropy_with_logits(logits = pred, labels = label) / FLAGS.
update_batch_size
    #xent()函数的返回值是一个平均损失,FLAGS.update_batch_size 代表当前学习的样本数,因此
    #tf.nn.softmax_cross_entropy_with_logits/FLAGS.update_batch_size 的当前计算结果就是当前
    #的平均损失
    #tf.nn.softmax_cross_entropy_with_logits()函数有 2 个输入。其中,logits 取当前预测值
    #在元学习模型中,tf.nn.softmax_cross_entropy_with_logits()函数是 softmax 激活函数和交叉
    #熵损失函数的结合体,用于计算多任务联合训练过程中的多分类损失,形成有全局指导意义的最
    #优化目标
```

3.5 元优化机制

3.5.1 元优化过程

元优化过程中调用的科学计算包主要来自 tensorflow.python.framework 工具包,该工具包包含元学习过程中的最优化机制(简称为元优化机制)。TensorFlow 集成了 Python 的常用开发框架 python.framework,这是一组用于简化和加速 Python 应用程序开发的库和工具。tensorflow.python.framework 工具包提供了一系列预定义的功能和结构,以便开发者能够快速构建、测试和维护应用程序。tensorflow.python.framework 工具包中的 ops(operations)模块是一个非常重要的组件,用于管理和监控系统运行状态。

本节所解释的元优化机制,主要包括元优化过程中科学计算包的 3 方面的机制,即导入机制、梯度下降机制和最大池化机制。现将代码分解为 3 个简单的小模块,以便读者轻松理解元优化机制的细节。

首先看元优化科学计算包的导入机制,核心代码如下。

```
"""
        #第一个模块导入元优化过程中所需的科学计算包
"""

from tensorflow.python.framework import ops
#从工具包 tensorflow.python.framework 导入 ops
#完成导入后,即可以使用 tensorflow.python.ops 模块

from tensorflow.python.ops import array_ops
#从 tensorflow.python.ops 模块导入 array_ops
#科学计算包 array_ops 中包含优化过程中的常用函数和类
#array 是排序的意思,即按照使用频率进行排序

from tensorflow.python.ops import gen_nn_ops
```

```
#从 tensorflow.python.ops 模块导入 gen_nn_ops
#科学计算包 gen_nn_ops 包含优化过程中的常用神经网络操作
#gen、nn 分别是 generate、neural network 的简写
```

3.5.2　拓展优化环境

元优化科学计算包的使用涉及对优化机制的理解。在解释元优化过程中的梯度下降机制之前，先拓展完成 Python＋PyCharm 的环境配置，此配置将在第 5 章用于 MAML 模型源代码的调试。感兴趣的读者，可以尝试在 Anaconda 下直接调试，那将是一个有趣的探索过程。

Python 的安装过程比较简单，打开如图 3-8 所示的官网。首先找到与计算机系统对应的安装包。单击 Downloads 按钮，可以看到 All releases（所有版本）、Source code（源代码）、Windows（Windows 系统对应的版本）、macOS（macOS 系统对应的版本）、Other Platforms（其他系统和平台对应的版本）、License（许可证，包括一些使用条款）、Alternative Implementations（包含 Python 安装配置的替代方式，主要是一些传统方式）选项，图 3-9 为操作系统是 Windows10 时的选择。

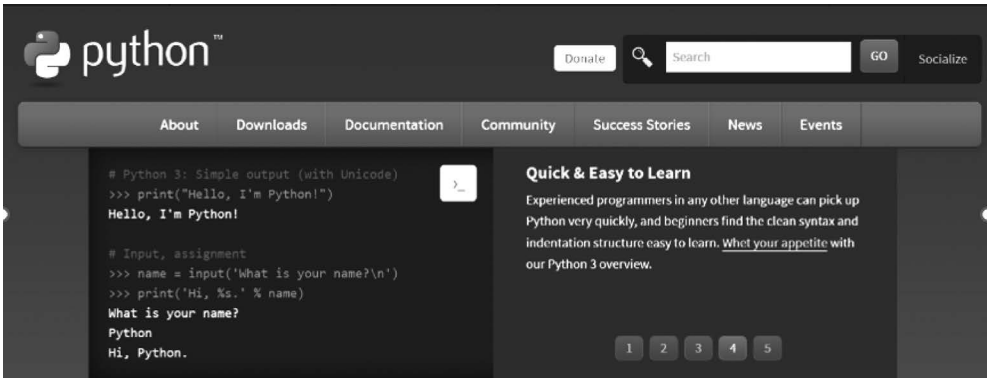

图 3-8　打开 Python 官网

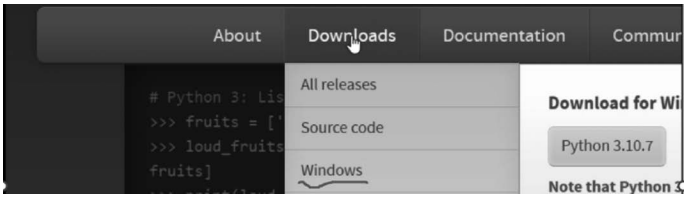

图 3-9　单击 Downloads 按钮并找到对应系统的选项

不建议下载最新版，因为最新版对应的工具包可能还没有发布。也不建议下载太老的版本，因为部分支持库不适用于旧版本的 Python。笔者的计算机操作系统是 64 位，选择的版本为 Python 3.7.4，单击对应的链接 Windows x86-64 executable installer，即可下载该版

本的安装包，预计 5 分钟左右完成，如图 3-10 所示。

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		68111671e5b2db4aef7b9ab01bf0f9be	23017663	SIG
XZ compressed source tarball	Source release		d33e4aae66097051c2eca45ee3604803	17131432	SIG
macOS 64-bit/32-bit installer	macOS	for Mac OS X 10.6 and later	6428b4fa7583daff1a442cba8cee08e6	34898416	SIG
macOS 64-bit installer	macOS	for OS X 10.9 and later	5dd605c38217a45773bf5e4a936b241f	28082845	SIG
Windows help file	Windows		d63999573a2c06b2ac56cade6b4f7cd2	8131761	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	9b00c8cf6d9ec0b9abe83184a40729a2	7504391	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	a702b4b0ad76debdb3043a583e563400	26680368	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	28cb1c608bbd73ae8e53a3bd351b4bd2	1362904	SIG
Windows x86 embeddable zip file	Windows		9fab3b81f8841879fda9413357413 9d8	6741626	SIG
Windows x86 executable installer	Windows		33cc602942a54446a3d6451476394789	25663848	SIG
Windows x86 web-based installer	Windows		1b670cfa5d317df82c30983ea371d87c	1324608	SIG

图 3-10　单击对应的按钮，下载最新版本的 Python 安装包

下载完成后的安装包文件名为 python-3.7.4-amd64.exe，双击该 EXE 文件，即可安装该版本的 Python，注意勾选 Add Python 3.7 to PATH 复选框，如图 3-11～图 3-13 所示。

图 3-11　最新版本的 Python 安装包已下载完成

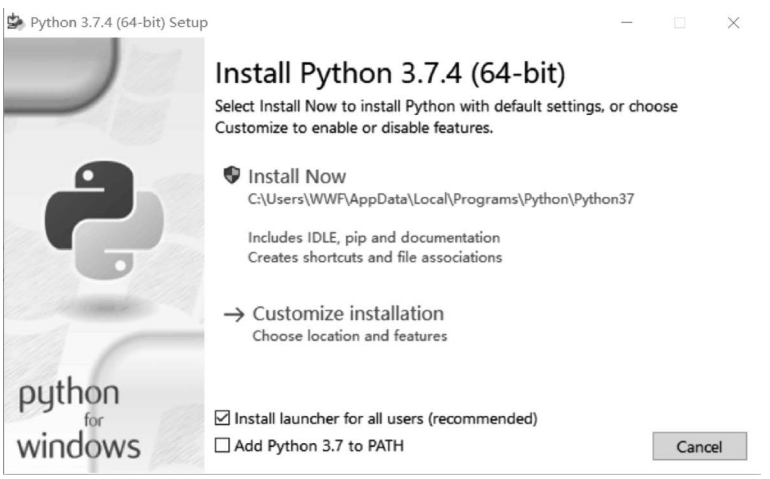

图 3-12　启动 Python 3.7.4 的安装

这里的 Customize installation 是自定义安装方式，可以选择安装路径和具体的安装内容。选择自定义安装方式，勾选所有 Optional Features 下的选项，如图 3-14 所示。

Install Now 按钮是快捷安装方式，在计算机中对应的直接安装路径（即快捷安装的位置）为 C:\Users\WWF\AppData\Local\Programs\Python\Python37，这是安装程序自动选择的路径。注意，其中默认包含了 IDLE（Python 的集成开发环境）、pip（Python 的包管理工具）及 Documentation（相关文件、文档）。这种安装方式还默认产生 Python 运行的快

图 3-13 安装正式开始之前，勾选 Add Python 3.7 to PATH 复选框

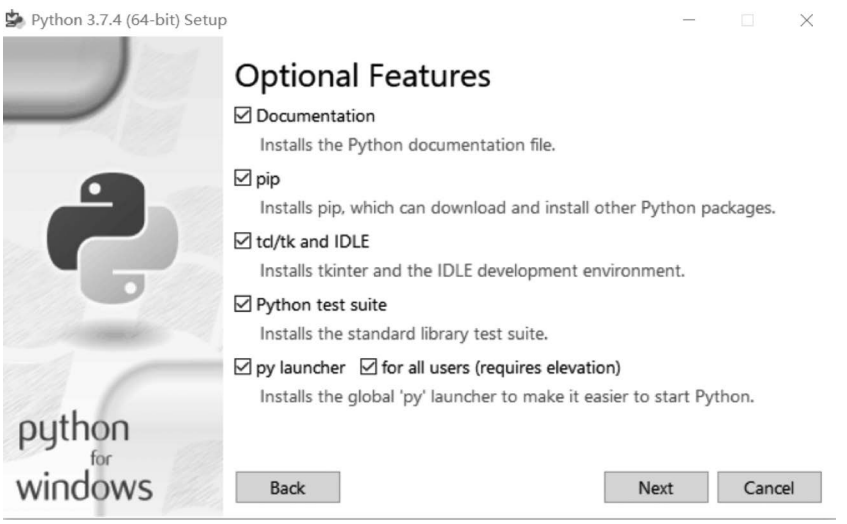

图 3-14　选择自定义安装，勾选所有“Optional Features”下的选项

捷方式(shortcuts)和文件打开方式的关联性(file associations)。

单击 Next 按钮，还可以看到其他高级选项，如图 3-15 所示。

此处保留已勾选的高级选项，然后单击 Browse 按钮更改安装路径，如图 3-16 所示。

单击“确定”按钮，安装路径已修改为 D:\Programfilesforai\Python，如图 3-17 所示。

单击 Install 按钮，弹出是否允许 Python 3.7.4 对该设备进行修改的确认信息，单击“I Agree”按钮，即可开始安装。由于只勾选了少数高级选项，安装过程不到一分钟即可完成，如图 3-18 所示。

安装完成后，出现安装成功的提示，如图 3-19 所示。

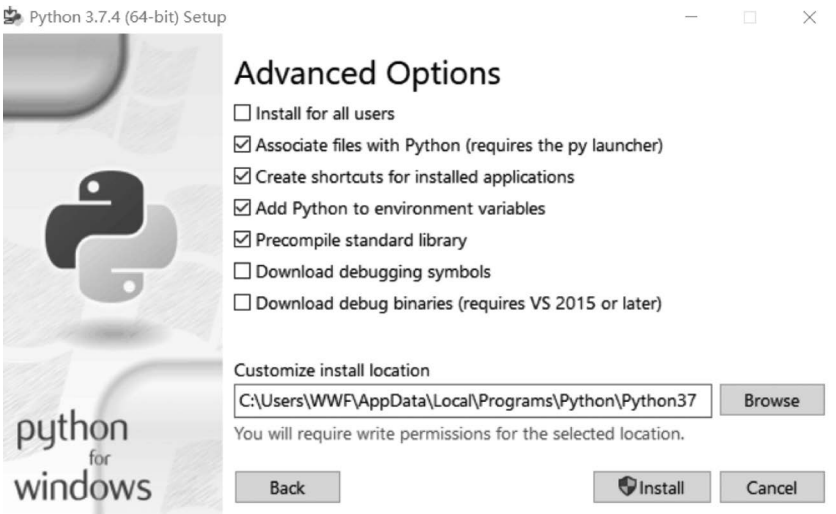

图 3-15　单击 Next 按钮后看到的其他高级选项

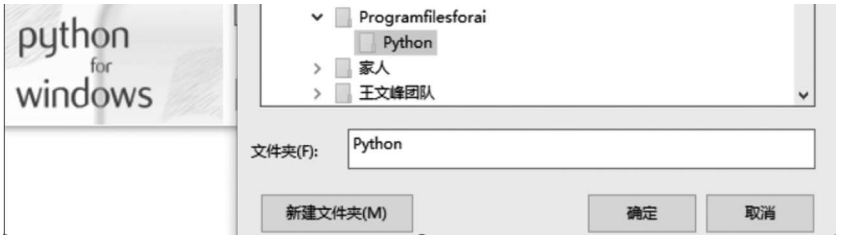

图 3-16　单击 Browse 按钮更改安装路径

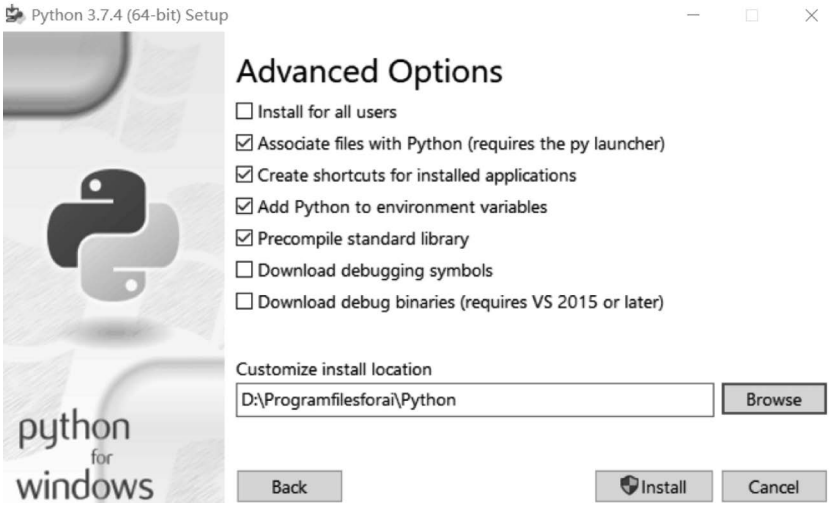

图 3-17　安装路径已修改为 D:\Program filesforai\Python

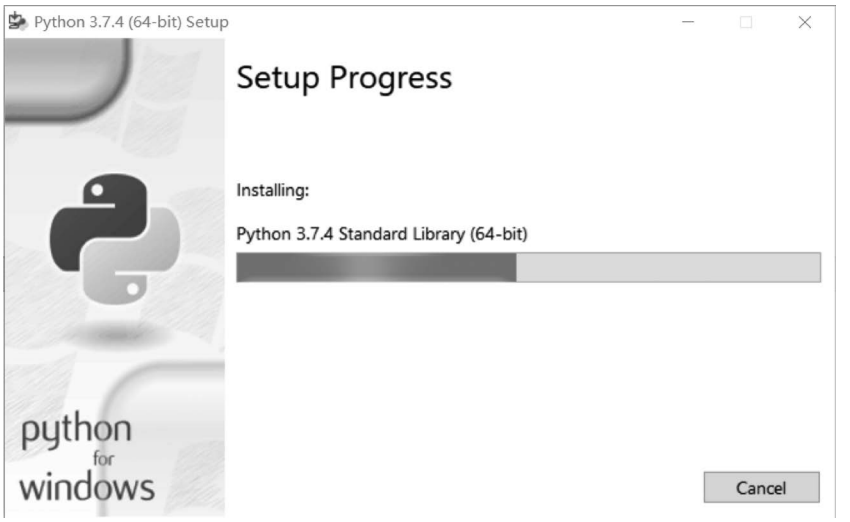

图 3-18　Python 3.7.4 的安装过程

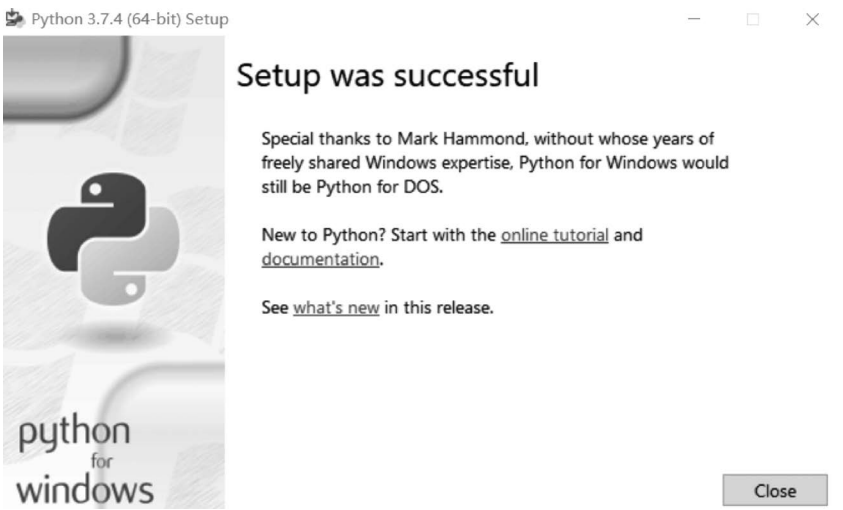

图 3-19　安装成功后出现的提示

该版本对 Python 初学者(new to Python)是友好的,提供在线教程(online tutorial)和辅助文档(documentation),读者可以选择对应的选项自行查阅学习。同时,上述安装成功界面也提示了扩展名为.py 的文件可以在 Python 内打开。读者还可以查看 what's New In Python 3.7 内的相关文档以了解该版本的新特性及如何在 Windows 系统上使用该版本(using Python on Windows),如图 3-20 所示。

需要注意的是,如果出现 Disable path length limit 提示信息,则是在提醒和引导去修改计算机配置(machine configuration),直接单击此按钮,弹出是否允许 Python 3.7.4 对该设备进行修改的对话框,单击"I Agree"按钮,即可完成修改。修改后的安装成功界面就没

图 3-20　Python 3.7 的新特性

有 Disable path length limit 提示了。单击 Close 按钮完成安装，接下来可以简单验证 Python 是否真的安装成功。

图 3-21　Python 3.7.4 自带的 IDLE

单击屏幕左下方的按钮，可以看到 IDLE(Python 3.7.4 自带的集成开发环境)，如图 3-21 所示。

单击 IDLE(Python 3.7 64 bit)按钮，启动 Python，在弹出的 Python 3.7.4 Shell 中输入一个简单的程序，然后按 Enter 键运行该程序。如果可以看到运行结果，说明 Python 3.7.4 安装成功。在 Python 3.7.4 Shell 中有 File(文件创建、打开、保存)、Edit(文件编辑)、Shell(脚本查看和调试)、Debug(调试)、Options(高级选项)、Window(* Shell IDLE 3.7.4)、Help(帮助)选项，如图 3-22 所示。

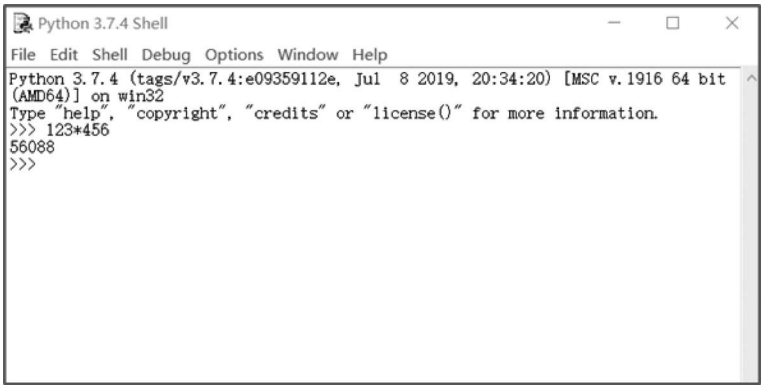

图 3-22　通过一个简单程序验证 Python 是否安装成功

也可以在 Python 命令行窗口进行验证。单击屏幕左下方“展开”旁的下拉箭头，可以看到 Python 3.7(64 bit)选项，单击此选项可以打开 Python 命令行窗口，如图 3-23～图 3-25 所示。

输入相同的简单程序，然后按 Enter 键运行该程序，可以看到相同的运行结果，如图 3-26 所示。

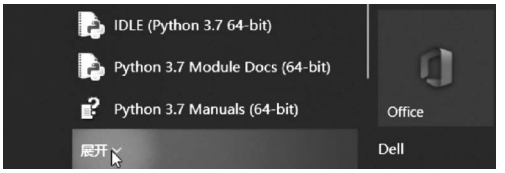

图 3-23　找到“展开”旁的下拉箭头并单击按钮

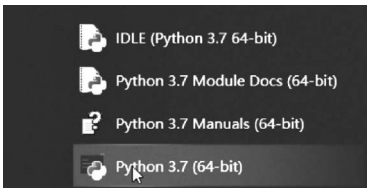

图 3-24　看到 Python 3.7 (64 bit)选项

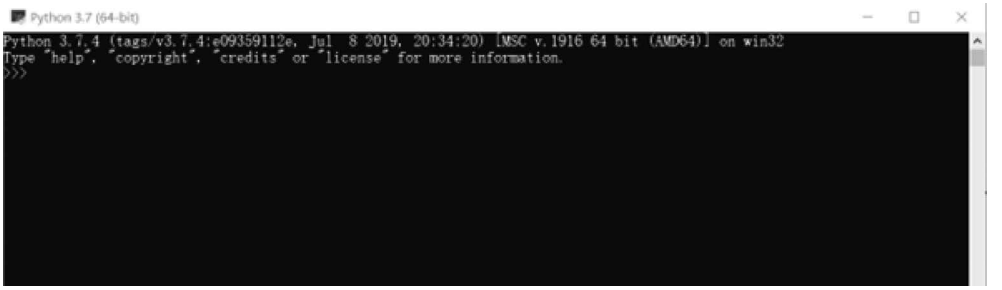

图 3-25　单击 Python 3.7 (64 bit)选项后，出现 Python 命令行窗口

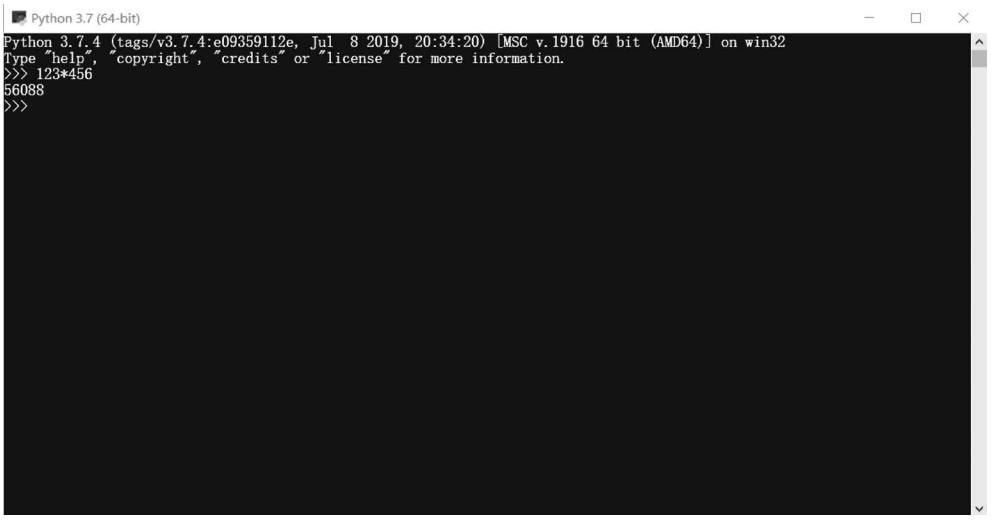

图 3-26　通过一个简单程序验证 Python 是否安装成功

元学习模型编程所需的科学计算包托管在 Anaconda 上，也可以直接通过 PyCharm 调用，所以需要先安装 Anaconda 和 PyCharm，已经在前两章完成 Anaconda 的安装配置。PyCharm 是 JetBrains(捷克的软件公司)开发的 Python 集成开发环境，有 Professional(专业版)和 Community(社区版)两种，其官网下载页面如图 3-27 所示。

此处选择下载免费的社区版(于 2022 年 2 月 2 日发布)，如图 3-28 和图 3-29 所示。

双击 EXE 文件 pycharm-community-2022.2.2，可以启动安装。在弹出的第一个窗口中单击 Next 按钮，在新弹出的窗口中单击 Browse 按钮以修改安装路径，继续单击 Next 按

钮，在弹出的窗口中勾选所有选项，再次单击 Next 按钮，如图 3-30～图 3-32 所示。

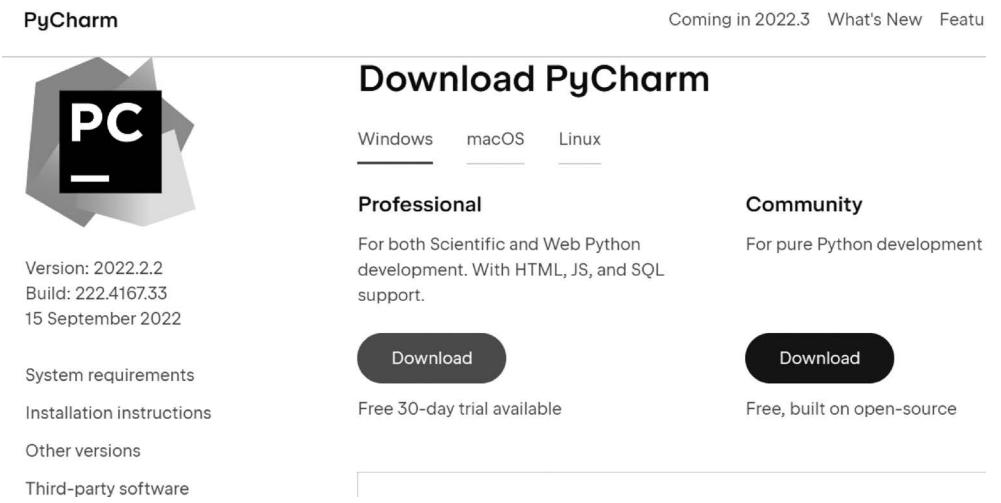

图 3-27　PyCharm 官网下载页面

图 3-28　社区版(PyCharm)正在下载

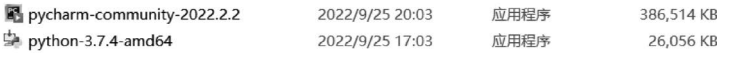

图 3-29　已下载的 PyCharm 安装包

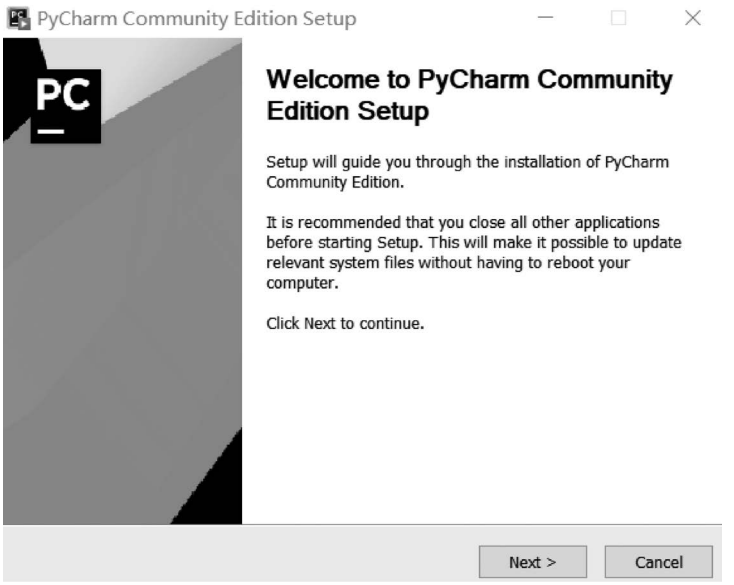

图 3-30　在弹出的第一窗口中单击 Next 按钮

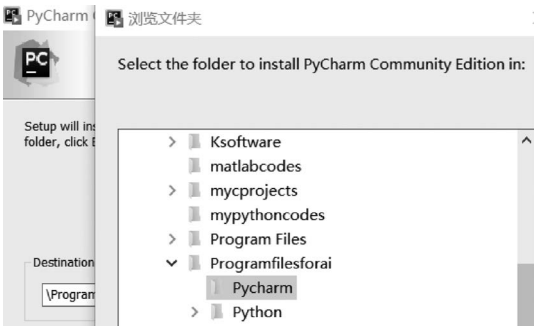

图 3-31　在新弹出的窗口中单击 Browse 按钮以修改安装路径

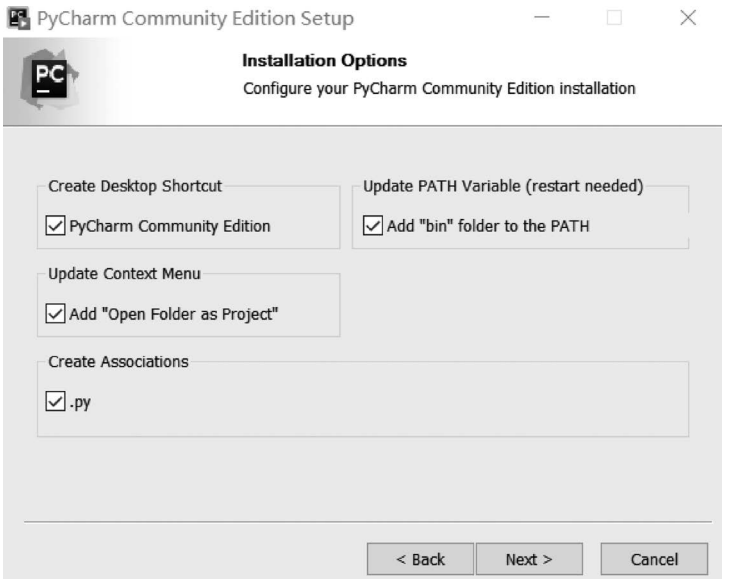

图 3-32　勾选所有安装选项

单击 Next 按钮，在弹出的窗口中单击 Install 按钮，即可完成安装，如图 3-33 所示。

安装完成后，需要重启计算机（选择 Reboot now 选项，并单击 Finish 按钮），如图 3-34 所示。

此时，可以在计算机桌面上看到 PyCharm 的图标，双击此图标即可打开 PyCharm。在弹出窗口中选择 Do not import settings 选项，并单击 OK 按钮即可启动 PyCharm，如图 3-35 所示。

成功启动 PyCharm 后出现的界面如图 3-36 所示。在左侧可以看到 4 个选项，分别为 Projects（项目）、Customize（自定义）、Plugins（插件，包括设置 PyCharm 操作界面语言为中文）、Learn PyCharm（学习 PyCharm）。其中，Projects 用于人工智能项目的开发，右侧对应有 New Project（创建新项目）、Open（打开已有项目）、Get from VCS（从之前配置的 GitHub 账号里获取该账号拥有的项目）3 个功能。

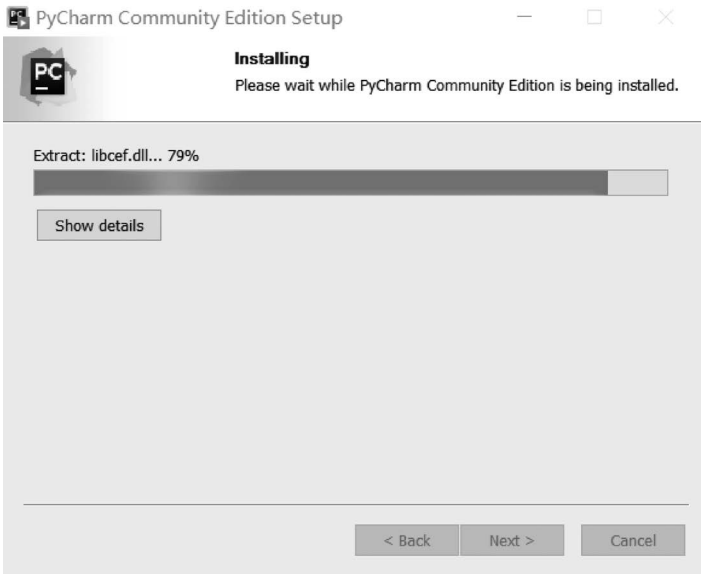

图 3-33　正在安装社区版 PyCharm

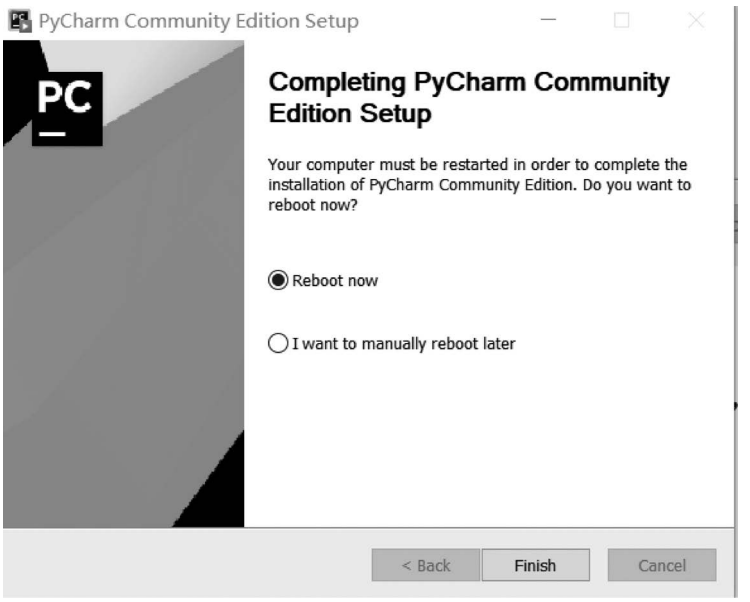

图 3-34　完成社区版 PyCharm 的安装

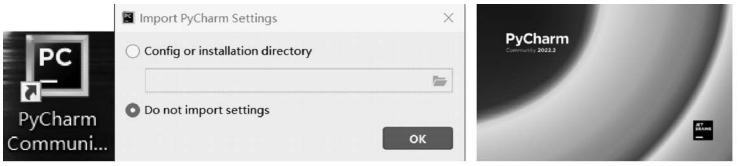

图 3-35　通过快捷方式启动 PyCharm

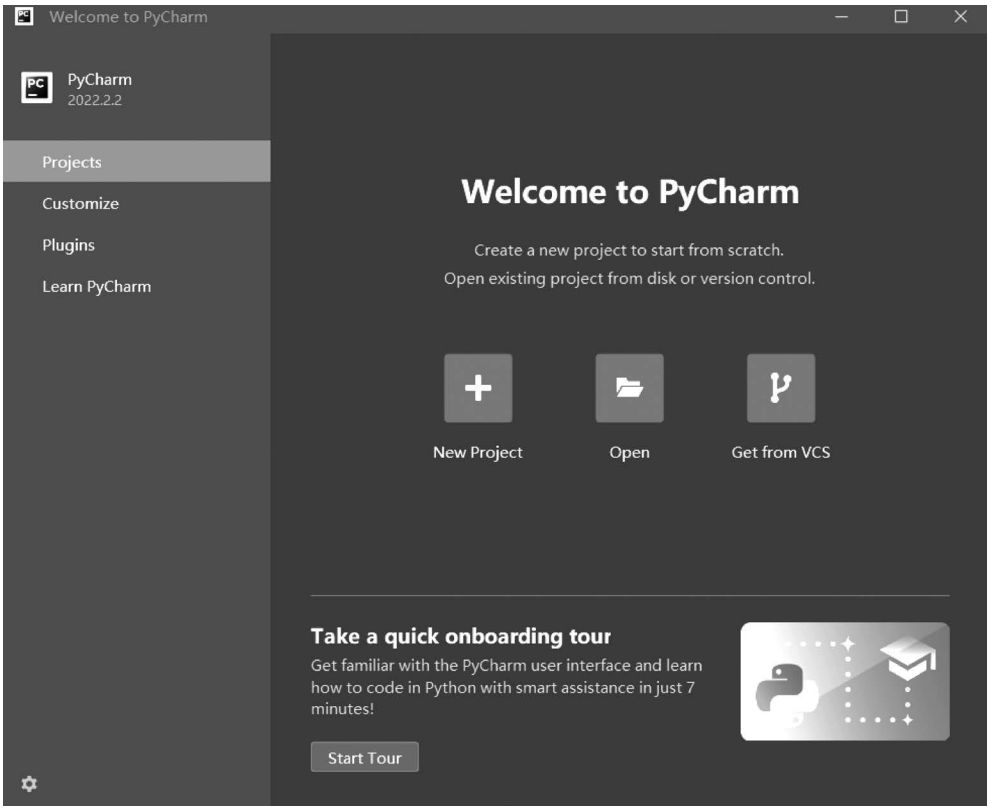

图 3-36　成功启动 PyCharm 后的界面

在右侧下方还可以看到 Take a quick onboarding tour 提示信息，意思是“你只需要花费 7 分钟，就可以熟悉 PyCharm 界面并学会在智能辅助下进行 Python 编程”。单击底部的 Start Tour 按钮，根据提示熟悉 PyCharm 界面。元学习代码调试所需的 PyCharm 配置将在第 4 章进行讲解。

3.5.3　最大池化过程

众所周知，TensorFlow 提供了比较完备的优化框架，但 TensorFlow 自带的科学计算包无法完全解决最优化过程中的二阶求导问题。因此，对过程建模而言，更为关键的还是梯度下降机制，其核心代码如下。

```
"""
#第二个模块注册元优化过程中所需的新 op
"""

@ops.RegisterGradient("MaxPoolGrad")
#注册一个新 op,并命名为 MaxPoolGrad,加入 ops,用于计算梯度
```

```
#顾名思义,主要用于计算最大池化梯度
#作为一个深度学习框架,TensorFlow 提供了大量的基本操作
#基本操作可任意组合并设计出比常用神经网络更强大的算法,使用优化技术的开发简单高效
#在元学习模型编程中,还会涉及一些不易实现的操作,此时就有必要注册新的 op
```

注册一个名为 MaxPoolGrad 的 op,主要是为了方便自定义梯度。换言之,MaxPoolGrad 作为一个新 op,可用于最大池化梯度的定义。该定义将有助于理解元优化过程中的最大池化机制,相关定义的核心代码如下。

```
"""
#第三个模块定义了用 op 和 grad 完成最大池化的过程
"""
def _MaxPoolGradGrad(op, grad):
#定义 _MaxPoolGradGrad()函数,顾名思义,两个 grad 连写表示连续两次求导
#该函的输入为 op 和 grad
#这里将计算出 3 个梯度值

    gradient = gen_nn_ops._max_pool_grad(op.inputs[0], op.outputs[0],
            grad, op.get_attr("ksize"), op.get_attr("strides"),
            padding = op.get_attr("padding"), data_format = op.get_attr("data_format"))
#反池化梯度 gradient 可以用 gen_nn_ops._max_pool_grad()函数计算
#其本质是计算最大池化的反向传播梯度,因此也称为反池化函数
#该函数有 7 个输入,依次为 op 的输入和输出模块、待求导梯度、op 自带的两个属性以及空间属
#性、数据格式

    gradgrad1 = array_ops.zeros(shape = array_ops.shape(op.inputs[1]), dtype = gradient.
dtype)
#第一个二阶导数 gradgrad1 是调用 array_ops.zeros()函数计算的
#array_ops.zeros()函数的输入为 shape 和 dtype,dtype 直接采用 gradient.dtype
#在 gradgrad1 中,shape 采用 array_ops.shape(op.inputs[1])

gradgrad2 = array_ops.zeros(shape = array_ops.shape(op.inputs[2]), dtype = gradient.
dtype)
#第二个二阶导数 gradgrad2 也可以调用 array_ops.zeros()函数计算
#array_ops.zeros()函数的输入为 shape 和 dtype,dtype 直接采用 gradient.dtype
#在 gradgrad2 中,shape 采用 array_ops.shape(op.inputs[2])

    return (gradient, gradgrad1, gradgrad2)
#_MaxPoolGradGrad()函数的返回值为 3 个梯度
#3 个梯度值依次为上述计算得到 gradient、gradgrad1 和 gradgrad2
```