第 1 章

图神经网络概述

本章将讲解图神经网络（Graph Neural Network，GNN）的基本概念和背景知识，内容包括：

- 什么是图神经网络
- 图神经网络的重要性
- 图神经网络与传统深度学习的区别

1.1 什么是图神经网络

1.1.1 图的基础知识

图的表示：图（Graph ）一般包括有向图和无向图。图的节点（Node）表示实体。图的边（Edge）表示节点之间的关系。通常可以使用邻接矩阵（Adjacency Matrix）来表示图。对于图神经网络来说，邻接矩阵是一个非常重要的概念。它可以将图的连通性形象地表示出来，节点与节点之间是否有边相连接，通过矩阵就可以知道。邻接矩阵利用二维矩阵表示图中各顶点之间的关系，对于有 n 个顶点的图来说，可以用 n 阶方阵来表示该图，其中矩阵元素 A_{ij} 表示从顶点 v_i 到 v_j 之间的边，A_{ij} 的大小表示边的权值。如果顶点 v_i 到 v_j 没有边，则可以将 A_{ij} 设置为 0 或者 ∞ 。图 1-1 所示为无向图及其邻接矩阵，图 1-2 所示为有向图及其邻接矩阵。

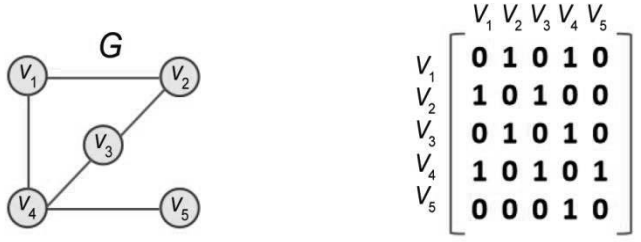

图 1-1　无向图及其邻接矩阵

图 1-2　有向图及其邻接矩阵

节点的度：表示与该节点相连的边的个数。

连通图（Connected Graph）：对于一个无向图，如果任意的节点 i 能够通过一些边到达节点 j，则称之为连通图。如果图中任意两个节点能够互相到达，则是强连通，否则是弱连通。对于无向图来说，若其是连通图，则一定是弱连通。对于有向图来说，如果任意两个节点之间都至少一个方向上有路径，则称该图是弱连通图；如果在两个方向上都有路径，则称该图是强连通图。

连通分量：无向图 G 的一个极大连通子图，称为 G 的一个连通分量（或连通分支）。连通图只有一个连通分量（Connected Component），即其自身。非连通的无向图有多个连通分量。

最短路径：两个节点直接相连或者通过其他节点可达的最近距离。

图直径：相隔最远的节点通过其他节点可达的最近距离。

1.1.2　图神经网络简介

图由节点和连接节点的边组成。它可以用来表示各种关系型数据，比如，社交网络中的用户与用户之间的关系、分子化学中原子与原子之间的连接、推荐系统中用户与物品的关联等。图神经网络（GNN）的主要目标是在这些图数据上进行各种任务，如节点分类、链接预测、图分类等。

图神经网络的输入一般是所有节点的起始特征向量和表示节点间连接关系的邻接矩阵。图神经网络的核心思想是基于节点的邻居关系进行信息传播和特征聚合。每个节点都有一个特征向量，表示节点的某些属性或特征。图神经网络通过迭代地更新节点的特征向量，使得节点能够逐渐获得来自邻居节点的信息。这种信息传播和聚合的过程允许模型捕捉节点之间的上下文关系，从而更好地理解图中的结构和模式。

以图 1-3 所示的图为例，首先考虑 A 节点，聚合操作会将当前节点 A 的邻居节点信息乘以系数聚合起来，加到节点 A 上作为 A 节点信息的补足信息，当作一次节点 A 的特征更新。

$$\text{邻居信息}N = a * (2,2,2,2,2) + b * (3,3,3,3,3) + c *(4,4,4,4,4)$$

$$A\text{的信息} = \sigma(W((1,1,1,1,1) + \alpha * N))$$

对图中的每个节点进行聚合操作，更新所有图节点的特征。通过不断交换邻域信息来更新节点特征，直到达到稳定均衡，此时所有节点的特征向量都包含其邻居节点的信息，然后就能利用这些信息来进行聚类、节点分类、链接预测等。这里举一个简单的例子。一次图节点聚合操作和权重学习可以理解为一层神经网络，后面再重复进行聚合、加权，就是多层迭代。一般 GNN 只要 3~5 层就可以学习到足够的信息。GNN 不会更新和改变输入图（Input Graph）的结构和连通性，所以我们可以用与输入图相同的邻接矩阵和相同数量的特征向量来描述 GNN 的输出图（Output Graph），输出图只是更新了每个节点的特征。

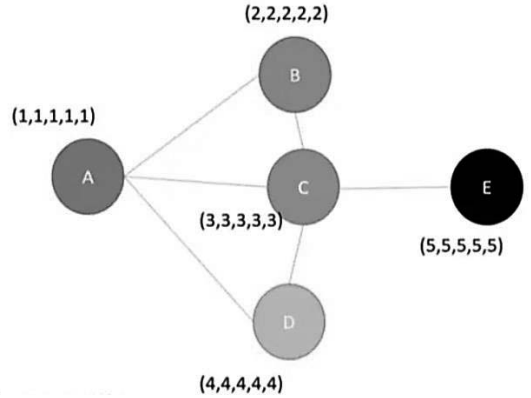

图 1-3　图神经网络示意图

不同的 GNN 模型可能在图卷积层的设计和信息聚合方式上有所不同，例如 GraphSAGE（Graph Sample and Aggregate，图采样和聚集）、GCN（Graph Convolutional Network，图卷积神经网络）、GAT（Graph Attention Network，图注意力网络）等。这些模型在不同的应用领域取得了显著的成果，使得处理图数据的机器学习变得更加高效和精确。总之，GNN 的出现极大地拓展了神经网络在复杂数据结构上的应用领域。

1.1.3　图神经网络的应用领域

下面讲解一下图神经网络（GNN）在不同领域中的应用场景，包括社交网络分析、推荐系统、生物信息学、交通网络优化等场景。

1. 社交网络分析

社交网络通常以图的形式表示，其中节点表示用户，边表示用户之间的关系。GNN 在社交网络分析中的应用包括：

- 节点分类：GNN 可用于识别社交网络中的用户类别，例如识别真实用户和垃圾用户。
- 链接预测：通过学习节点之间的关系，GNN 可以预测社交网络中未来可能的连接，如友谊关系或信息传播路径。
- 社群检测：GNN 可帮助发现社交网络中的社群或群体，以便更好地理解用户群体的行为和互动模式。

2. 推荐系统

推荐系统通过分析用户–物品关系图（如用户–电影、用户–商品）来提供个性化推荐。GNN 在推荐系统中的应用包括：

- 用户和物品嵌入学习：GNN 可以学习用户和物品的嵌入，从而更好地捕捉用户的兴趣和物品的特征，用于个性化推荐。
- 推荐路径分析：GNN 可以分析用户与物品之间的交互路径，以识别潜在的用户兴趣演化和转换路径。

3. 生物信息学

在生物信息学中，GNN 的应用范围广泛，包括：

- 蛋白质相互作用预测：GNN 可分析蛋白质之间的相互作用网络，帮助预测蛋白质之间的相互作用，从而揭示生物学过程中的关键信息。
- 药物发现：GNN 可用于学习化合物结构和生物活性之间的关系，从而加速药物发现过程。
- 基因表达分析：GNN 可分析基因调控网络，帮助理解基因表达的调控机制和相互关系。

4. 交通网络优化

在交通网络中，GNN 的应用包括：

- 交通流量预测：GNN 可用于分析道路网络中的交通流量数据，帮助预测交通拥堵和优化交通管理。
- 路线规划：GNN 可用于优化最佳路线，考虑实时交通情况和道路网络拓扑。
- 交通信号优化：GNN 可帮助优化交通信号控制，以提高交通流畅度和减少交通拥堵。

这些应用示例突显了图神经网络（GNN）在多个领域中的广泛适用性。通过学习图数据的表示和关系，GNN 提供了一种强大的方式来解决复杂的图数据分析和优化问题，为各种应用提供了新的机会和方法。

1.2 图神经网络的重要性

近年来，神经网络的成功推动了模式识别和数据挖掘的研究。许多机器学习任务，如目标检测、机器翻译和语音识别，曾经严重依赖手工制作的特征工程来提取信息特征集，最近已经被各种端到端深度学习范式彻底改变，例如卷积神经网络（CNN）、循环神经网络（RNN）和自动编码器（Auto Encoder）。深度学习在许多领域的成功部分归功于快速发展的计算资源（例如 GPU）、大量训练数据的可用性，以及深度学习从欧几里得数据（例如图像、文本和视频）中提取潜在表征的有效性。以图像数据为例，可以将图像表示为欧几里得空间中的规则网格。卷积神经网络能够利用图像数据的移位不变性、局部连通性和组合性。因此，卷积神经网络可以提取与整个数据集共享的、局部有意义的特征，用于各种图像分析。

虽然深度学习有效地捕获了欧几里得数据的隐藏模式，但越来越多的应用程序将数据以图形的形式表示。例如，在电子商务中，基于图形的学习系统可以利用用户和产品之间的交互来做出高度准确的推荐。在化学中，分子被建模为图形，它们的生物活性需要被确定以用于药物发现。在引文网络中，文章通过引文相互链接，它们需要被分类到不同的组中。图的复杂性给现有的机器学习算法带来了巨大的挑战。由于图可以是不规则的，一个图可能有不同大小的无序节点，来自一个图的节点可能有不同数量的邻居，导致一些重要的操作（如卷积）在图像域很容易计算，但很难应用到图域。此外，现有机器学习算法的一个核心假设是实例彼此独立。这种假设不再适用于图数据，因为每个实例（节点）通过各种类型的链接（如引用、交互）与其他实例（节点）相关。

当我们处理非结构化数据（如文本、图像等）时，可以使用神经网络来学习它们的特征表示。然而，对于更加复杂的数据结构，例如图数据，传统的神经网络模型往往显得无力。在这样的背景下，图神经网络应运而生。它是一类特殊的神经网络，旨在处理图结构数据并学习节点之间的关系。

1.3　图神经网络与传统深度学习的区别

传统深度学习是指在深度学习领域的早期阶段使用的一系列基本技术和方法。这些方法通常包括：

- 人工神经网络（Artificial Neural Network，ANN）：传统深度学习的核心是人工神经网络，通常是基于多层感知器（Multilayer Perceptron，MLP）的模型。这些模型由输入层、隐藏层和输出层组成，每个神经元都与前后层的神经元相连接，通过权重来调整连接的强度。
- 卷积神经网络（Convolutional Neural Network，CNN）：针对图像处理任务的深度学习应用，引入了卷积层和池化层，以捕捉空间关系和降低数据维度。
- 循环神经网络（Recurrent Neural Network，RNN）：用于处理序列数据的深度学习模型，具有记忆性能。
- 反向传播（Backpropagation）算法：反向传播是一种用于训练神经网络的优化算法，通过计算梯度来更新网络参数，以最小化损失函数。
- 激活函数（Activation Function）：激活函数用于引入非线性性质，常见的激活函数包括 Sigmoid、ReLU（Rectified Linear Unit）和 Tanh 等。
- 损失函数（Loss Function）：损失函数用于衡量模型的预测与实际标签之间的差距，常见的损失函数包括均方误差（Mean Squared Error）和交叉熵损失（Cross-Entropy Loss）等。
- 优化算法（Optimization Algorithm）：用于更新神经网络参数的优化算法，如随机梯度下降（Stochastic Gradient Descent）及其变种。
- 批处理（Batch Processing）：将训练数据划分为小批量来进行模型参数的更新，有助于训练的稳定性和加速。
- 正则化（Regularization）：为了避免过拟合，可以使用正则化技术，如 L1 正则化和 L2 正则化，来约束模型参数的大小。
- 初始化策略（Initialization Technique）：合适的权重初始化对于训练深度神经网络非常重要，例如 Xavier 初始化和 He 初始化。

这些传统深度学习技术为现代深度学习的发展奠定了基础，但现代深度学习已经发展到了更高级的阶段，包括使用卷积神经网络进行图像识别、使用循环神经网络进行自然语言处理、使用生成对抗网络进行图像生成等。在深度学习领域的应用也不断演进，包括自动驾驶、医疗诊断、自然语言处理和语音识别等。

1.3.1 传统深度学习模型

下面将详细介绍卷积神经网络、循环神经网络、生成对抗网络这三种经典的传统深度学习模型。

1. 卷积神经网络

卷积神经网络是一种带有卷积结构的深度神经网络，卷积结构可以减少深层网络占用的内存量，其有三个关键的操作：局部感受野、权值共享和池化层，有效地减少了网络的参数个数，缓解了模型的过拟合问题。

卷积神经网络的结构组成如下。

- 输入层（Input Layer）：用于数据的输入。
- 卷积层（Convolution Layer）：卷积神经网络中每层卷积层由若干卷积单元组成，每个卷积单元参数通过反向传播算法优化得到。卷积运算的目的是提取输入的不同特征，每一层卷积层智能提取一些低级的特征，如边缘、线条和角等层级，更多层的网络能从低级特征中迭代提取更复杂的特征。
- 激活函数（Activation Function）：将卷积层的输出非线性化，最常用的激活函数是 ReLU。不被记作单独层数。
- 池化层（Pooling Layer）：减少图像特征（Feature Map）的空间尺寸，减少训练参数数量。
- 全连接层（Fully Connected Layer）：把所有局部特征结合变成全局特征，一般用来计算每一类的得分，起到分类器的作用，一般都使用 softmax 激活函数量化最终的输出。
- 输出层（Output Layer）：输出最终结果。

卷积神经网络（CNN）在图像处理、计算机视觉等领域有着广泛的应用，以下是一些常见的应用场景。

- 图像分类：CNN 可以对图像进行分类，如将一幅照片分为人、动物、建筑等不同类别。
- 目标检测：CNN 可以在图像中检测出特定的目标，如行人、车辆等。
- 人脸识别：CNN 可以对人脸进行识别和匹配，如在人脸门禁系统中的应用。
- 图像生成：CNN 可以生成逼真的图像，如 GAN（Generative Adversarial Network，生成对抗网络）模型可以生成逼真的人脸、风景等图像。
- 自动驾驶：CNN 可以对路况进行识别和分析，以实现自动驾驶等功能。
- 医学影像分析：CNN 可以对医学影像进行分析和诊断，如对 CT、MRI 等影像进行病灶检测和分类。
- 自然语言处理：CNN 可以对文本进行分类和情感分析，如对新闻文章进行分类和情感分析等。

总之，卷积神经网络在图像处理、计算机视觉、自然语言处理等领域有着广泛的应用，随着深度学习技术的不断发展和进步，其应用场景也将不断扩大和深化。

2. 循环神经网络

循环神经网络（RNN）是一种常用的神经网络结构，它源自 1982 年由 Saratha Sathasivam 提出的霍普菲尔德网络。

RNN 背后的想法是利用顺序信息。在传统的神经网络中，假设所有输入（和输出）彼此独立。但对于许多任务而言，这是不合理的。如果想预测句子中的下一个单词，那么最好知道它前面有哪些单词。RNN 被称为“循环”，这是因为它们对序列的每个元素执行相同的任务，输出取决于先前的计算。考虑 RNN 的另一种方式是它有一个“记忆”，可以捕获到目前为止计算的信息。图 1-4 所示为典型的 RNN 网络在 t 时刻展开的样子。

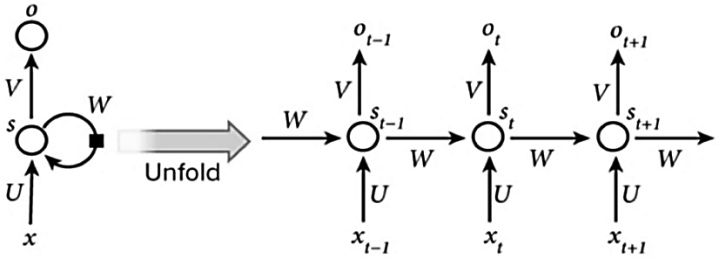

图 1-4　循环神经网络

其中，x_t 是输入层的输入；s_t 是隐藏层的输出，s_0 是计算第一个隐藏层所需要的，通常初始化为全零；o_t 是输出层的输出。

从图中可以看出，RNN 网络的关键一点是 s_t 值不仅取决于 x_t，还取决于 s_{t-1}。

这里需要注意：

- 将隐藏的状态 s_t 看作网络的记忆，它捕获有关所有先前时间步骤中发生的事件的信息。步骤输出 o_t 根据时间 t 的记忆计算。它在实践中有点复杂，因为 s_t 通常无法从太多时间步骤中捕获信息。
- 与在每层使用不同参数的传统深度神经网络不同，RNN 共享相同的参数（所有步骤的 U、V、W）。这反映了在每个步骤执行相同任务的事实，只是使用不同的输入，这大大减少了我们需要学习的参数总数。

3. 生成对抗网络

生成对抗网络（GAN）包含两个模型，一个是生成模型（Generative Model），另一个是判别模型（Discriminative Model）。判别模型的任务是判断给定的实例看起来是自然真实的还是人为伪造的（真实实例来源于数据集，伪造实例来源于生成模型）。

如图 1-5 所示为生成对抗模型的模型框架，训练 GAN 有两个部分。

（1）鉴别器（Discriminator）：在这个阶段，网络只进行前向传播，不进行反向传播。判别器在真实数据上训练 n 个 epoch（轮数），看看是否可以正确地将它们预测为真实数据。此外，在这个阶段，鉴别器还接受了生成器生成的假数据的训练，看看是否可以正确地将它们预测为假数据。

（2）生成器（Generator）：生成模型的任务是生成看起来自然真实的、和原始数据相似的实

例。在鉴别器空闲时训练生成器。在通过生成器生成的假数据训练判别器之后，可以得到它的预测并使用结果来训练生成器，使生成的数据更接近真实数据，从而试图欺骗判别器。将上述方法重复几个 epoch，然后检查虚假数据是否看起来是真实的。如果它看起来可以接受，则停止训练，否则允许它再继续几个 epoch。

这样，两个模型就在相互竞争，在博弈论意义上是对抗性的，在玩零和游戏。在这种情况下，零和意味着当判别器成功识别真假样本时，它会获得奖励或不需要更改模型参数，而生成器会因模型参数的大量更新而受到惩罚。或者，当生成器欺骗判别器时，它会得到奖励，或者不需要更改模型参数，但判别器会受到惩罚并更新其模型参数。

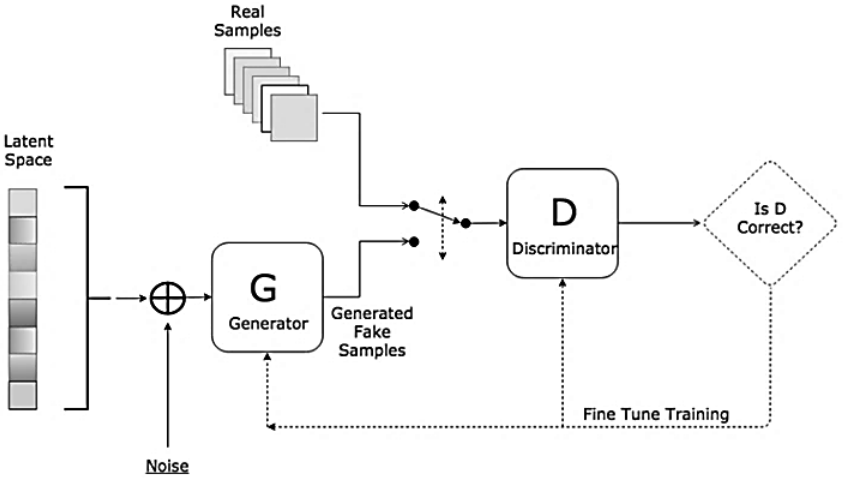

图 1-5　生成对抗模型

1.3.2　图神经网络与传统深度学习的区别

在处理图结构数据时，图神经网络（GNN）与传统深度学习方法之间存在一些重要的区别。

1. 在数据结构上

传统深度学习方法通常处理向量或矩阵形式的数据，如文本、图像等。这些数据结构中缺少显式的节点和边的概念。而图神经网络专门用于处理图结构数据，其中数据由节点和边组成，节点表示实体，边表示实体之间的关系。图神经网络能够捕捉节点之间的关系，使得它们适用于处理社交网络、分子结构等图数据。

2. 在信息传播和聚合上

传统深度学习方法通过层叠的全连接层、卷积层等来提取特征。信息在网络中以固定的方式传递，没有明确的信息传播和聚合过程。图神经网络则以节点为中心，通过迭代的信息传播和聚合过程更新节点的特征表示。每一轮迭代中，节点会考虑其邻居节点的特征，从而捕捉节点之间的关系。

3. 在节点关系建模上

在传统深度学习中，节点之间的关系往往隐含在数据的特征中，模型需要从数据中自行学习这

些关系。图神经网络直接建模节点之间的关系，通过信息传播和聚合来捕捉这些关系。这使得图神经网络能够更好地理解节点之间的上下文信息。

4. 迭代性质

传统深度学习一般是前馈的，每一层的输出直接作为下一层的输入，没有显式的迭代过程。图神经网络具有迭代性质，信息在节点之间传播和聚合，每一轮迭代会更新节点的特征。这使得图神经网络能够逐步聚焦于节点的局部和全局信息。

总的来说，图神经网络是为了更好地处理图结构数据而设计的，通过显式的信息传播和聚合过程，能够捕获节点之间的关系并提取有用的特征。传统深度学习方法适用于处理向量和矩阵形式的数据，而图神经网络则在处理图数据方面表现出色。这两者在处理不同类型的数据和问题上具有不同的优势。

第 2 章

PyTorch 开发环境搭建

本章将讲解图神经网络开发环境的搭建，内容包括：

- Anaconda 的安装和配置
- PyCharm 的安装和配置
- PyTorch 的安装和配置
- PyTorch Geometric 的安装和配置

2.1 Anaconda 的安装和配置

Python 是一种面向对象的解释型计算机程序设计语言，其使用具有跨平台的特点，可以在 Linux、macOS 以及 Windows 系统中搭建环境并使用。其编写的代码在不同平台上运行时，几乎不需要做较大的改动，使用者无不受益于它的便捷性。

此外，Python 的强大之处在于它的应用范围之广，遍及人工智能、科学计算、Web 开发、系统运维、大数据及云计算、金融、游戏开发等。实现其强大功能的前提，就是 Python 具有数量庞大且功能相对完善的标准库和第三方库。通过对库的引用，能够实现对不同领域业务的开发。然而，正是由于库的数量庞大，对于管理这些库以及对库进行及时维护成为既重要但复杂度又高的事情。

Anaconda 是一种科学计算环境，可以便捷获取包且对包能够进行管理，同时可以对环境统一管理。Anaconda 包含 Conda、Python 在内的超过 180 个科学包及其依赖项。

Anaconda 的官方网址为 https://www.anaconda.com/download，从官网下载官方安装程序 Anaconda3-2023.07-2-Windows-x86_64.exe，双击运行，并按照操作提示进行安装。当成功安装后，会在桌面上生成 Anaconda Navigator (Anaconda3)的应用程序图标，双击打开后如图 2-1 所示，这个就是 Anaconda 的桌面应用程序。

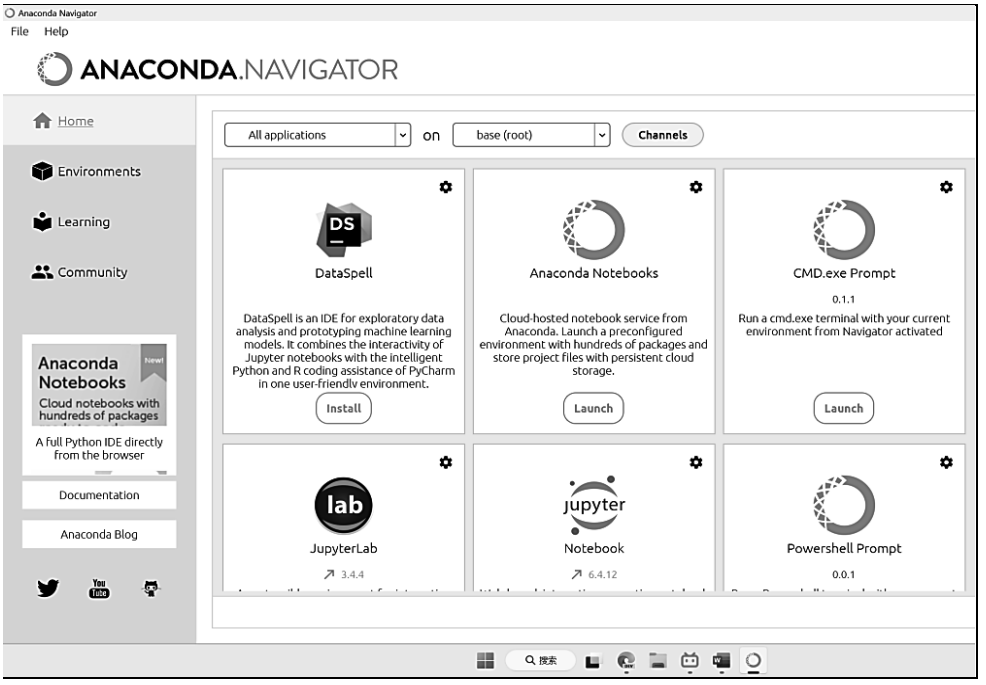

图 2-1　Anaconda 的桌面应用程序

使用 Anaconda 的桌面应用程序，可以很方便地为每一个 Python 项目创建分离的虚拟环境，为不同的项目加载不同版本的包，并对项目下的包及其依赖关系进行管理。

如图 2-2 所示，单击程序中的 Environments，可以查看当前计算机中存在的虚拟环境，单击下方的 Create 按钮可以创建新的虚拟环境，输入环境名称并选择 3.9.17 版本的 Python，单击 Create 按钮即可创建相应 Python 版本的虚拟环境。

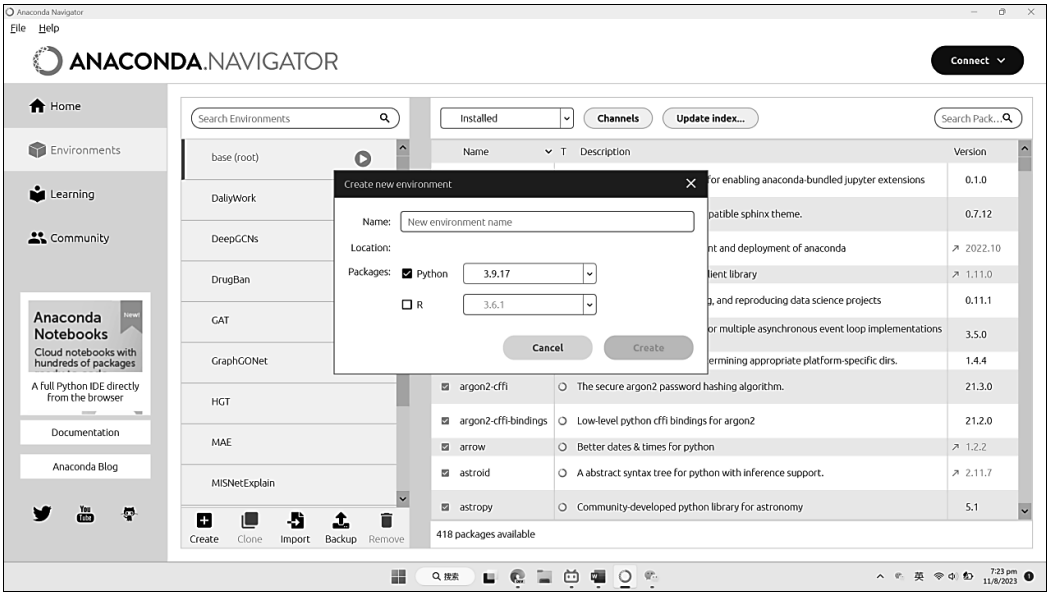

图 2-2　创建虚拟环境

2.2 PyCharm 的安装和配置

Python 中自带了一个 IDE，在安装好 Python 后即可进行代码的编写，但是这样编写程序效率很低，在实际应用中一般使用其他集成 IDE 作为 Python 的开发环境，如 PyCharm、Visual Studio Code、Eclipse with PyDev 等。在这里，我们使用的是 PyCharm 集成开发环境。PyCharm 的官方下载地址为 https://www.jetbrains.com/pycharm/download，选择页面下方的社区版进行下载，并按照安装向导的提示进行安装。

PyCharm 是基于项目进行管理的，可以新建一个项目或者打开一个已经存在的项目，并对其进行相应的设置。首先打开 PyCharm，单击 File→New Project…菜单新建项目，如图 2-3 所示。或者单击 File→Open…菜单打开已存在的项目，如图 2-4 所示。

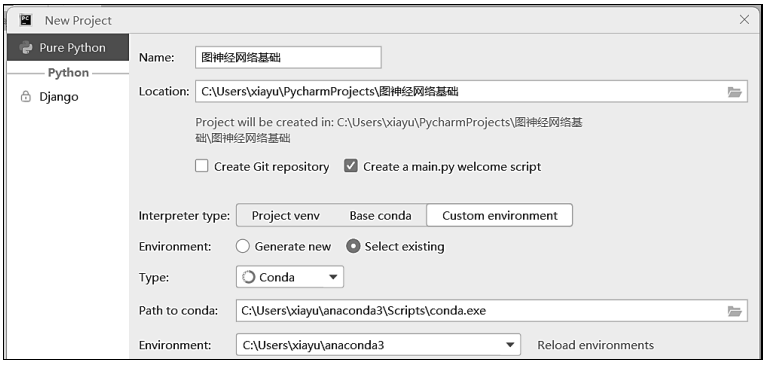

图 2-3 创建 PyCharm 项目

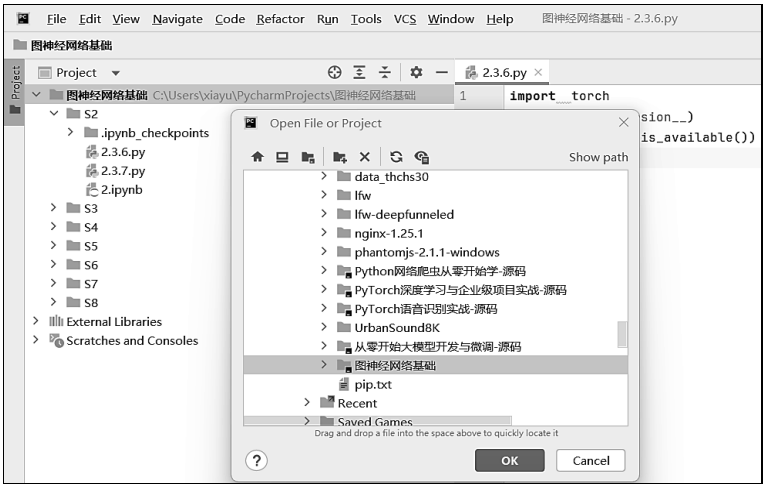

图 2-4 打开已经存在的项目

选择项目地址后，再配置一下项目的解释器。打开项目后，单击 File→Settings…菜单，打开 Settings 窗口，如图 2-5 所示。单击 Settings 窗口右上方的 Add Interpreter 添加解释器，按图 2-6 选择添加本地解释器，Conda 可执行文件是位于 Anaconda 安装地址下 Scripts 文件夹中的 conda.exe，选择好后单击 Load Environments 按钮，然后就可以选择已经创建好的 Conda 环境或者创建新的环境了。

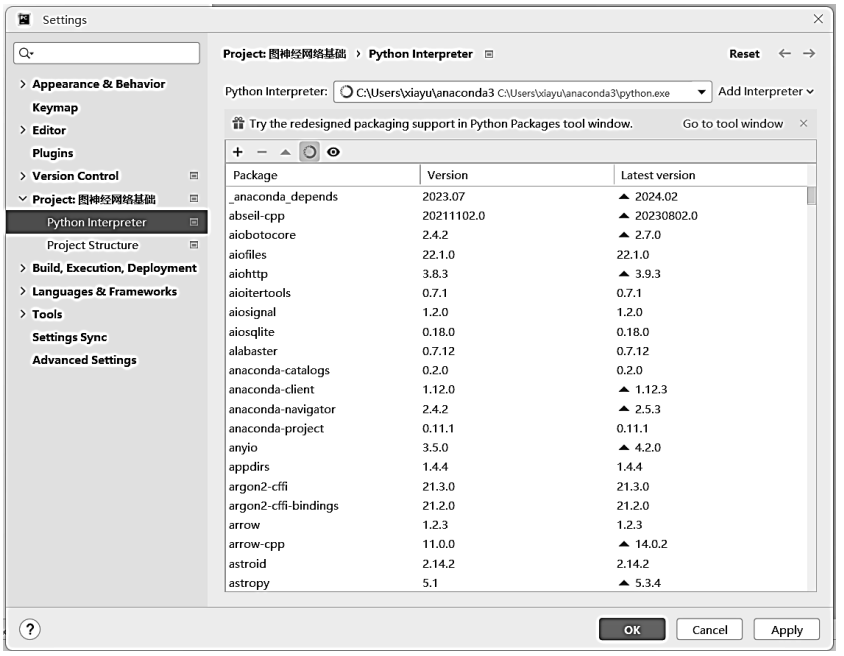

图 2-5　链接 Conda 虚拟环境

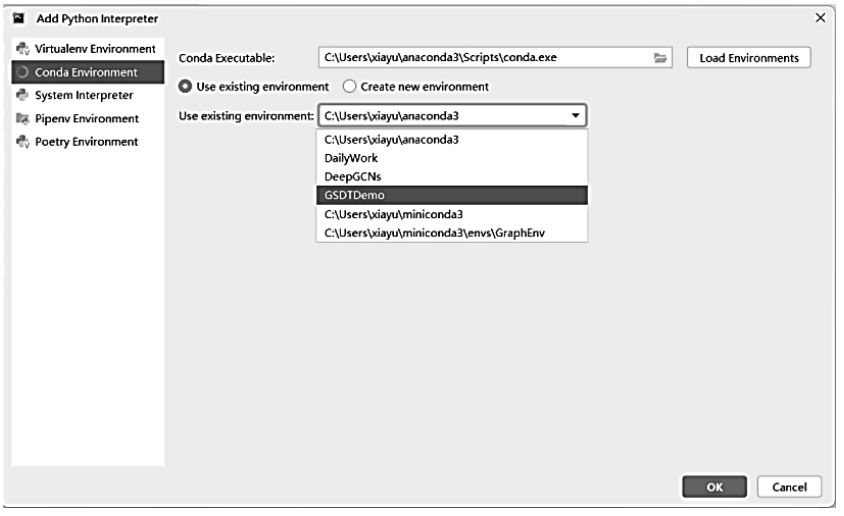

图 2-6　链接 Conda 虚拟环境

单击 OK 按钮回到上一级界面 Settings 窗口后，再单击 OK 按钮即可完成项目的配置。

2.3　PyTorch Geometric 的安装和配置

在安装好 Anaconda 包管理工具和 PyCharm 开发环境后，我们已经能够进行 Python 代码的编写，现在需要进行 PyTorch 深度学习环境的搭建。进行通用深度学习环境的搭建，需要安装 Navida CUDA

与 cuDNN，方便深度学习应用进行 GPU 调用。

2.3.1 查看系统支持的 CUDA 版本

打开 Navida 控制面板，单击“帮助”→“系统信息”→“组件”，查看计算机的 CUDA 版本，如图 2-7 所示。

图 2-7 查看系统 CUDA 版本

2.3.2 下载最新的 Navida 显卡驱动

打开 Navida 官方网站（https://www.nvidia.com/download/index.aspx?lang=en-us），选择自己的计算机对应的显卡和操作系统，下载最新的显卡驱动，如图 2-8 所示。

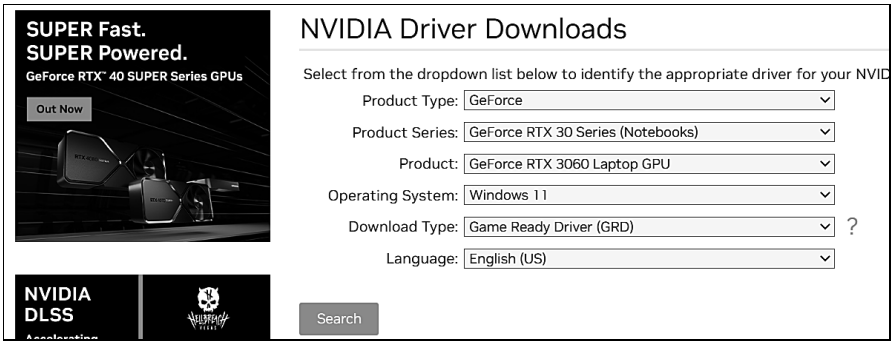

图 2-8 下载 Navida 显卡驱动

2.3.3 下载 CUDA Toolkit

打开 CUDA 工具网址（https://developer.nvidia.com/cuda-toolkit-archive），选择自己的计算机支

持的 CUDA 版本，进行 CUDA 工具的安装，比如作者选择 11.8.0 版本，如图 2-9 和图 2-10 所示。

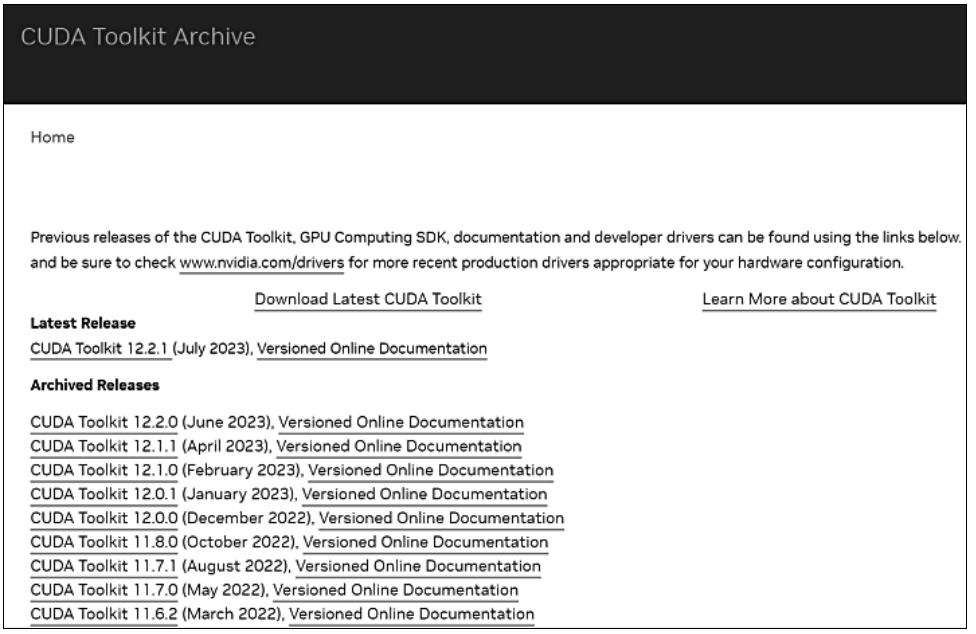

图 2-9　下载自己的计算机对应 CUDA 版本的 CUDA 工具

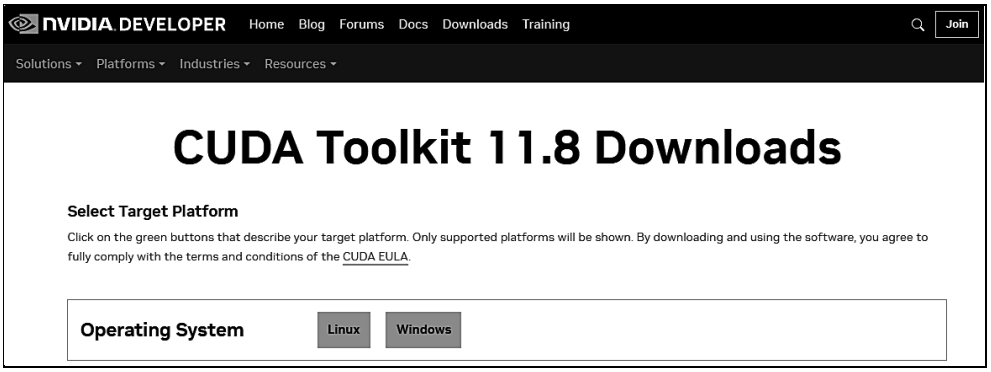

图 2-10　选择自己的计算机对应的操作系统进行安装

安装完成后，打开 CMD 窗口，输入命令 nvcc--version，可以查看 CUDA 是否安装成功，如图 2-11 所示。

```
PS C:\Users\xiayu> nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:41:10_Pacific_Daylight_Time_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
```

图 2-11　检查 CUDA 工具是否安装成功

2.3.4 cuDNN 的安装

cuDNN 下载地址为 https://developer.nvidia.com/rdp/cudnn-download，下载 cuDNN 需要注册 Navida 账户，再选择对应的 CUDA 版本进行下载，如图 2-12 所示。

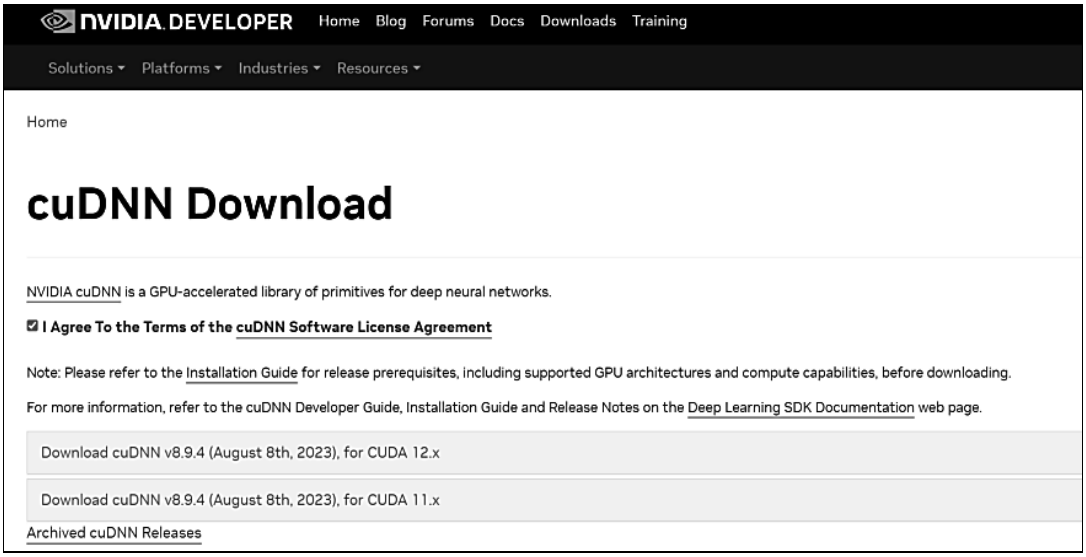

图 2-12 下载 cuDNN

下载完成后解压缩，会得到三个子文件夹，如图 2-13 所示。

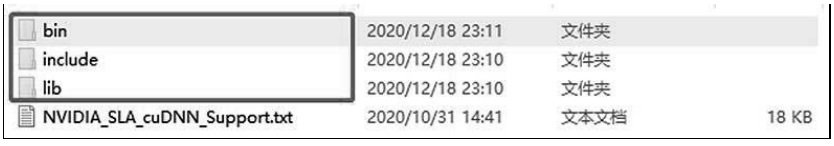

图 2-13 cuDNN 解压缩后的目录

CUDA 的默认安装路径为 C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v 版本号。将 cuDNN 三个文件夹的内容分别复制到 CUDA 主目录下对应的文件夹里面，如图 2-14 所示。

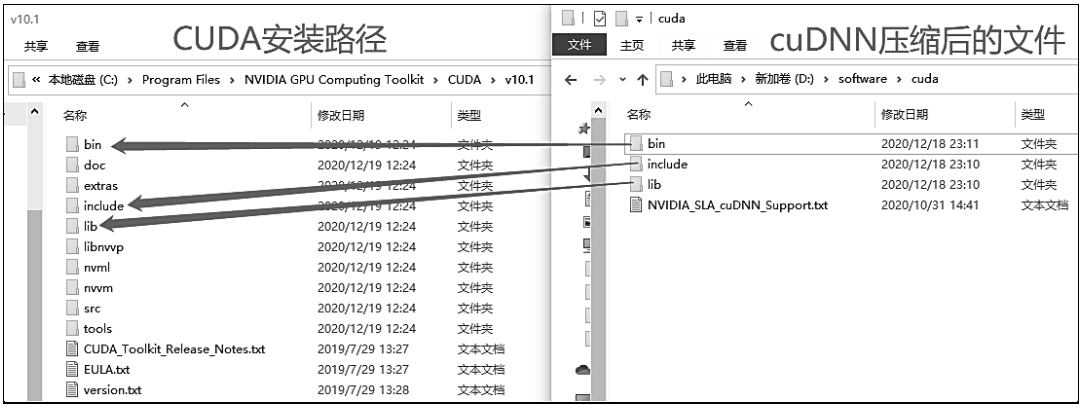

图 2-14 将 cuDNN 中的文件复制到 CUDA 目录下

2.3.5　安装 PyTorch 框架虚拟环境

打开 PyTorch 的官方网址 https://pytorch.org/get-started/previous-versions/，在历史版本中找到我们需要的版本，复制 conda 命令安装的代码，如图 2-15 所示。

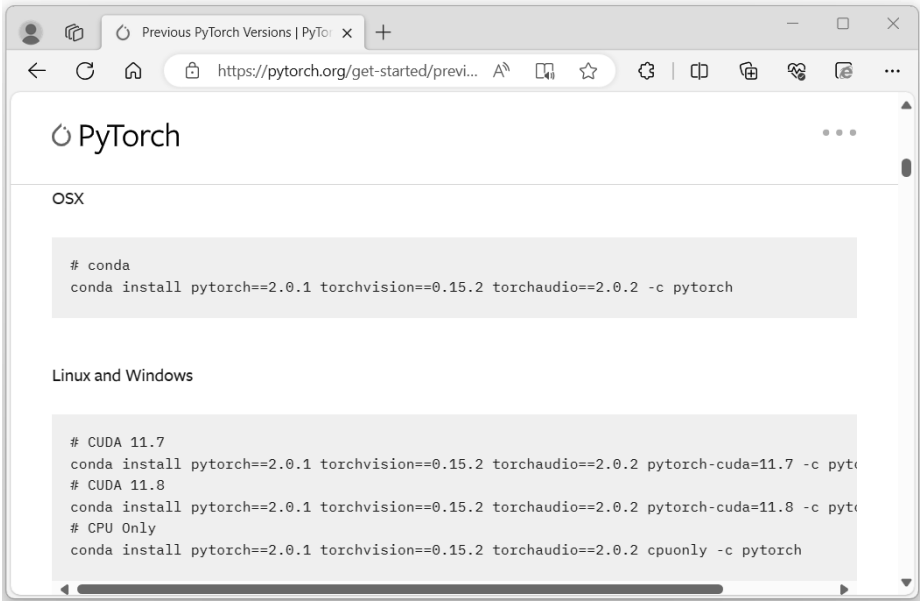

图 2-15　在 PyTorch 官网选择所需的版本

这里我们以安装 PyTorch 2.0.1 版本、CUDA 11.8 版本为例，代码为：

```
conda install pytorch==2.0.1 torchvision==0.15.2 torchaudio==2.0.2
pytorch-cuda=11.8 -c pytorch -c nvidia
```

获得代码后，打开 PyCharm 中之前新建的项目，单击左侧下方终端按钮，使用如下命令创建并激活虚拟环境（其中 GraphEnv 是我们自定义的环境名）：

```
conda create -n GraphEnv python=3.9
conda activate  GraphEnv
conda install pytorch==2.0.1 torchvision==0.15.2 torchaudio==2.0.2
pytorch-cuda=11.8 -c pytorch -c nvidia
#安装其他工具包
deactivate  #Exit virtual environment
```

然后输入复制的代码并按 Enter 键确认，即可下载对应的 PyTorch 框架，如图 2-16 所示。虚拟环境创建好后，可以在 PyCharm 中为本书项目设置 Python Interpreter，具体操作为 File→Setting…→Python Interpreter→Add Interpreter→Add Local Interpreter→Conda Environment→Load Environment→Use existing environment→GraphEnv（这个 GraphEnv 就是我们前面创建的虚拟环境。针对不同项目的环境要求，我们可以相应地创建不同的虚拟环境）。

在后面各章运行示例项目时，如果对 PyTorch 版本及其他依赖包版本有特殊要求，均可以按前面讲解的方法，创建并激活新的虚拟环境来支持。

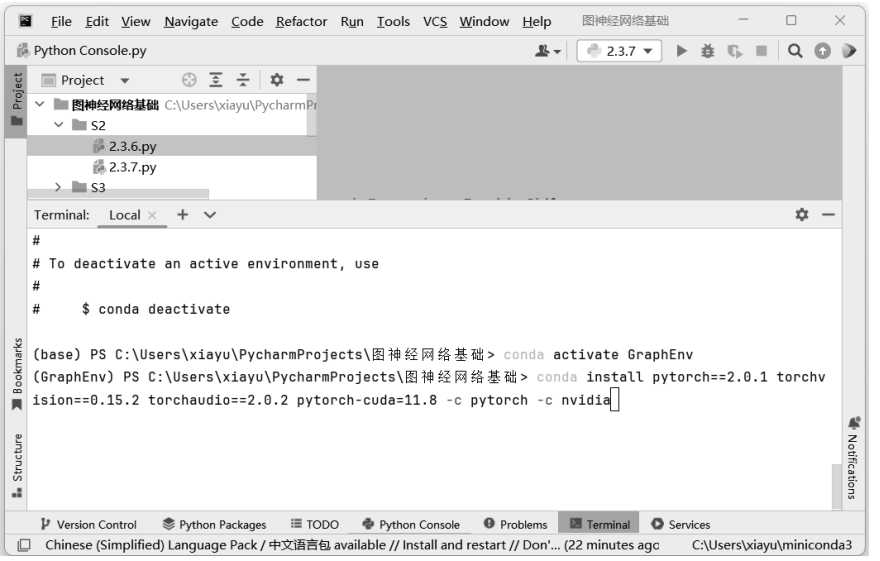

图 2-16　在 PyCharm 中打开终端，激活虚拟环境后安装库文件

2.3.6　检查 PyTorch 框架的安装

新建一个2.3.6.py文件，运行以下代码。如果能够成功运行并返回torch版本与True，说明PyTorch框架安装成功并调用了 GPU。

```
import  torch
print(torch.__version__)
print(torch.cuda.is_available())
```

运行代码，返回结果说明 PyTorch 安装成功，如图 2-17 所示。

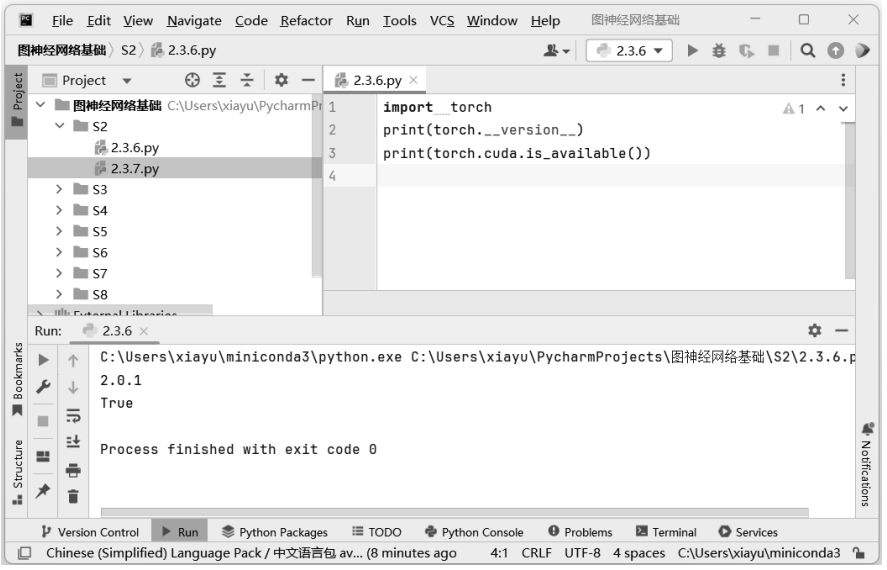

图 2-17　返回结果说明 PyTorch 安装成功

2.3.7　安装图神经网络库

torch 的图神经网络需要安装额外模块。首先在 https://pytorch-geometric.com/whl/中选择对应的 torch 与 CUDA 版本。如图 2-18 所示。比如，单击 torch-2.0.1+cu118 链接打开下载页面。

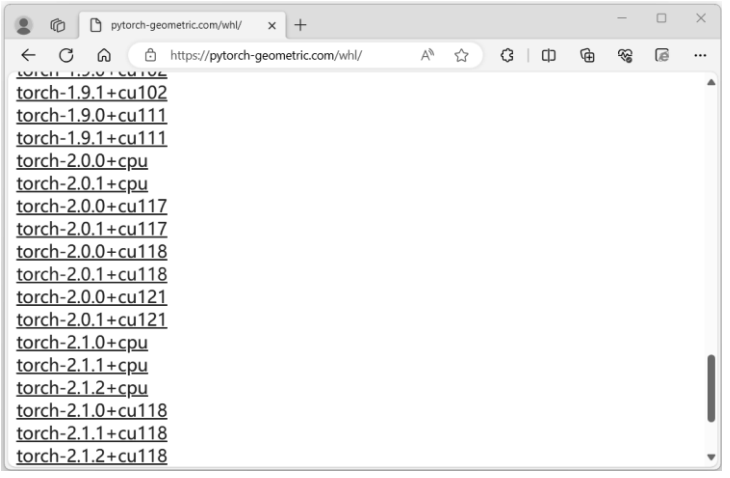

图 2-18　选择对应的 torch 版本与 CUDA 版本

然后选择 torch_cluster、torch_scatter、torch_sparse、torch_spline_conv 四个与项目对应的操作系统与 Python 版本的.whl 文件，并将它们下载下来，如图 2-19 所示。

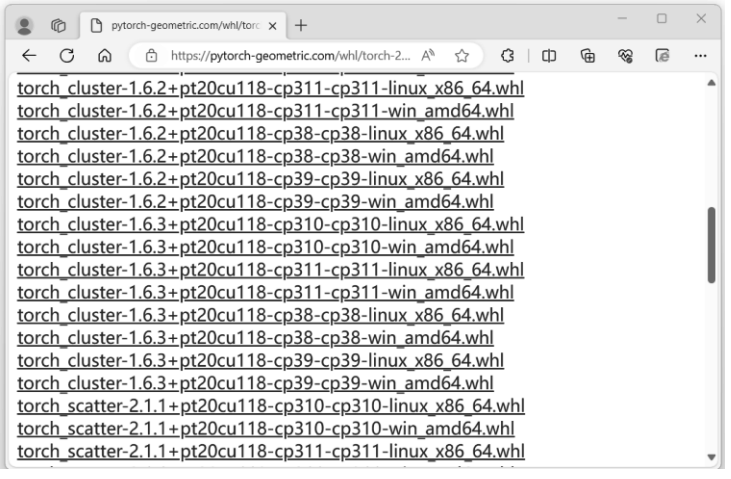

图 2-19　选择对应 Python 版本、CUDA 版本和操作系统版本的库文件

下载完成后，将 4 个.whl 文件放置到项目文件夹下，然后打开 PyCharm 项目，在终端中使用 pip 命令安装本地文件，如下所示：

```
pip install torch_cluster-1.6.3+pt20cu118-cp39-cp39-win_arm64.whl
pip install torch_scatter-2.1.2+pt20cu118-cp39-cp39-win_amd64.whl
pip install torch_sparse-0.6.18+pt20cu118-cp39-cp39-win_amd64.whl
pip install torch_spline_conv-1.2.2+pt20cu118-cp39-cp39-linux_x86_64.whl
```

4 个相关库都安装完成后，再在终端中输入命令 pip install torch-geometric，即可安装图神经网络库 torch-geometric。

以上库文件安装完成后，接下来测试一下图神经网络库是否安装成功。新建一个.py 文件，复制以下代码并运行：

```
import torch_sparse
import torch_scatter
import torch_cluster
import torch_spline_conv
import torch_geometric
import torch
print("torch_sparse: ", torch_sparse.__version__)
print("torch_scatter: ", torch_scatter.__version__)
print("torch_cluster: ", torch_cluster.__version__)
print("torch_spline_conv: ", torch_spline_conv.__version__)
print("torch_geometric: ", torch_geometric.__version__)
print("torch: ", torch.__version__)
```

如果上面的代码运行成功，就表明图神经网络库安装成功，如图 2-20 所示。

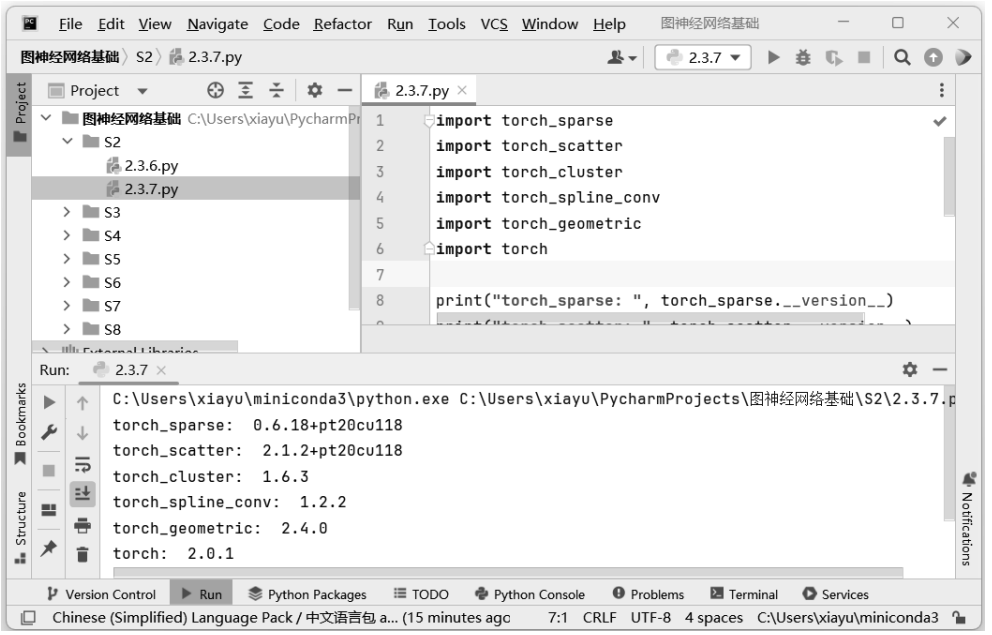

图 2-20　代码运行结果

2.3.8　使用 Jupyter Notebook 运行代码

2.1 节安装 Anaconda 时，也安装了 Jupyter Notebook。Jupyter Notebook 提供了一个 Web 代码运行环境，用户可以在其页面上编写代码、运行代码、查看结果并可视化数据。本书示例源码建议在 Jupyter Notebook 中运行，运行方法是在管理员终端执行命令：jupyter notebook，执行命令之后，在

终端中将会显示一系列 Notebook 的服务器信息，如图 2-21 所示。

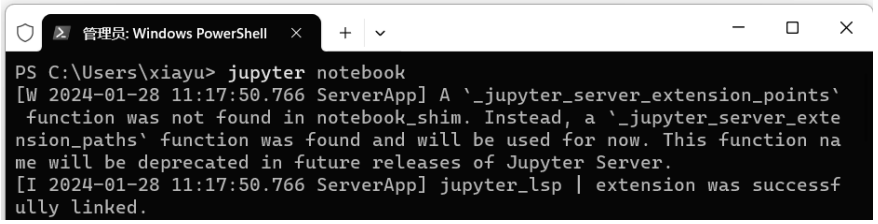

图 2-21　运行 Notebook 服务器

同时将会自动启动系统默认的浏览器，打开 Jupyter Notebook 运行环境，界面如图 2-22 所示。使用 Notebook 运行环境时，不能关闭图 2-22 所示的终端管理员窗口，否则 Notebook 服务会被关闭。如果 Notebook 服务关闭了，可以重新运行 jupyter notebook 命令，打开 Notebook 服务。

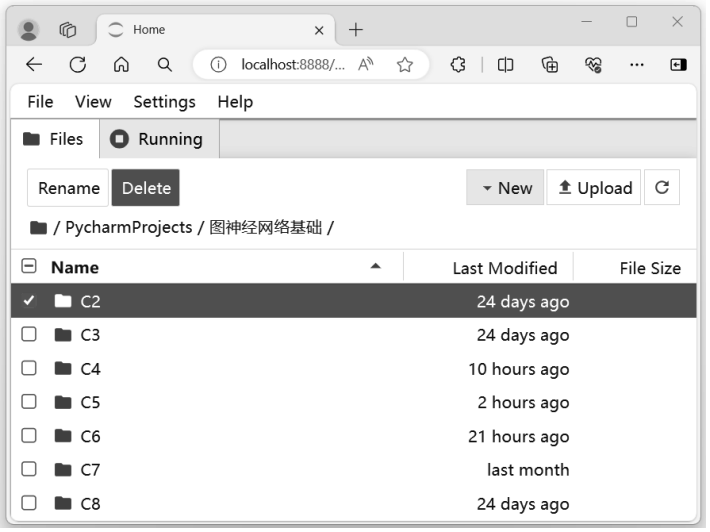

图 2-22　浏览器中的 Jupyter Notebook 界面

如果要打开并运行 Notebook 代码文件，比如打开第 2 章的 2.ipynb，可在如图 2-22 所示的界面中，按目录层次找到这个示例文件，双击打开并逐个运行代码段，界面如图 2-23 所示。

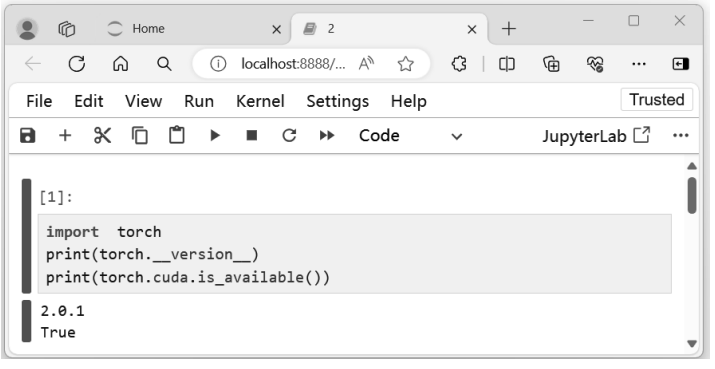

图 2-23　在 Notebook 界面中打开示例文件