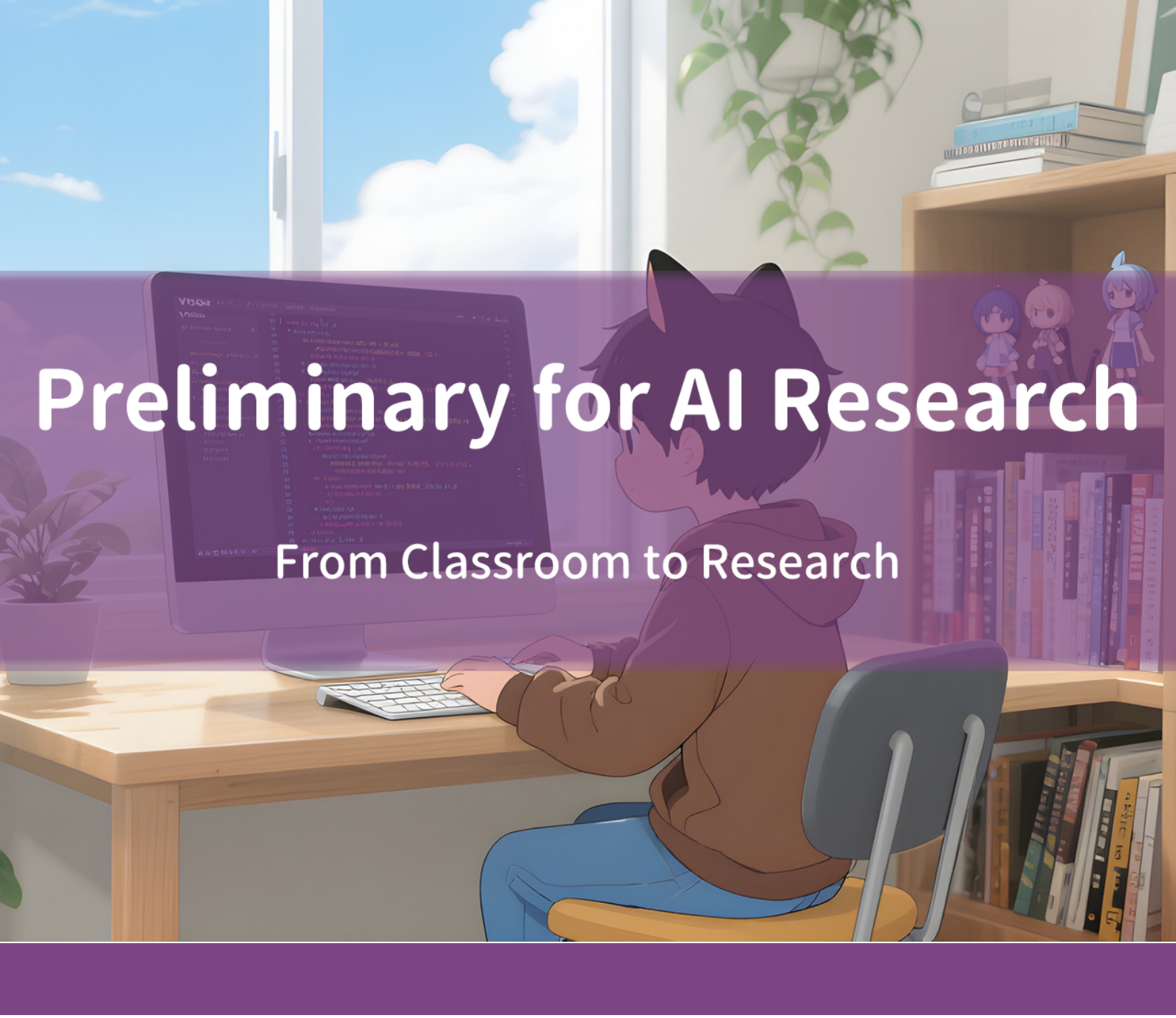

【新手友好】AI 科研常用知识

实践中常用的编程、深度学习、机器学习、数学

作者：Xi Wang (Hytidel)

版本：2025.09.10

$$e^{\pi i} + 1 = 0$$

From Classroom to Research.

前言

本课程旨在介绍从本科生的课堂学习到 AI 科研的衔接知识，所涉及的知识大多是各 AI 领域的科研中通用的。此外，本课程还作为笔者的后续课程（敬请期待）的先修课，因此会补充介绍计算机视觉 (Computer Vision, CV) 和计算机图形学 (Computer Graphics, CG) 中需用到的知识。

本课程的主要内容如下：5章介绍了 AI 科研中常用的数学知识（应少尽少），它本应被放在第 1 章，但笔者知道这样会直接劝退绝大部分人，故将其放在最后一章，但前面的章节仍会引用到5章的知识，请读者需要时自行查阅。后续的章节也大致是难度逐渐上升的顺序安排。具体地，1章介绍了常用的编程知识，主要是 Python 和常用的库；2章介绍了常用的可视化和绘图方法，让你能亲手画出论文中的图；3章着重介绍和补充 AI 科研实践中常用的深度学习知识，尤其是 *PyTorch* 框架的使用，并附有代码实现；4章介绍了一些常用的机器学习知识，它们会在某些领域的科研中出现。此外，每章还配有简单的实验或例题作为示例和习题。

值得一提的是，本课程基本只列出了每个板块在实践（即论文、代码）中常用的基础知识 (preliminary)。作为一门主打“广度”的课，不可避免地会损失一些“深度”。若希望深入学习某个板块，我们也附上了优质的学习资源的链接，请读者自行学习。

虽然本课程的定位是“喂饭”教程，但仍假设读者有本科的基础的数学课（高等数学/数学分析、线性代数/高等代数、概率论与数理统计）、计算机课（*C* 语言的语法、*Python* 的语法、面向对象基础）和深度学习（如动手学深度学习[1]）的基础。对于过于基础的知识，我们也附上了优质的学习资源的链接，请读者自行学习。

本课程的配套讲解视频为[2]（建议关注、一键三连），GitHub 仓库为[3]（建议 star）。

[1] https://courses.d2l.ai/zh-v2/

[2] https://space.bilibili.com/382329676/lists/6307119?type=season

[3] https://github.com/Hytidel/preliminary_for_ai_research

目录

第一章 编程基础

1.1 环境

1.1.1 查询显卡信息

1.1.1.1 CUDA

Compute Unified Device Architecture (CUDA) 是 NVIDIA 提出的并行计算平台和编程模型，可让开发者用 C/C++/Python 等语言操作 GPU。GPU 内有很多个 CUDA 核心，计算高度并行。

1.1.1.2 nvidia-smi

在终端（cmd, powershell, bash, shell 等）运行

```
nvidia-smi
```

输出如下：

```
(no_pain_diffusion) PS C:\Users\Hytidel> nvidia-smi
Mon Sep 8 09:07:24 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 566.07            Driver Version: 566.07     CUDA Version: 12.7  |
|-----------------------------------------+------------------------+----------------------+
| GPU Name                 Driver-Model | Bus-Id      Disp.A | Volatile Uncorr. ECC |
| Fan Temp  Perf        Pwr:Usage/Cap |       Memory-Usage | GPU-Util Compute M. |
|                                     |                        |             MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA GeForce RTX 4060 ... WDDM | 00000000:01:00.0 Off |           N/A |
| N/A  50C   P8            1W /  80W |     72MiB /  8188MiB |    0%     Default |
|                                     |                        |               N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI       PID   Type   Process name                          GPU Memory |
|       ID   ID                                                                  Usage     |
|=========================================================================================|
|   0   N/A  N/A    12392   C+G   D:\TencentQQ\QQ.exe                     N/A     |
+-----------------------------------------------------------------------------------------+
```

上方为 head 信息和 GPU 概览表，其中的重要参数：

1. `Driver Version: 566.07` 表示 NVIDIA 驱动器 (Driver) 版本为 566.07；
2. `CUDA Version: 12.7` 表示该驱动**最高**支持的 CUDA Runtime 版本；
3. `GPU` 表示显卡的编号（0-indexed，即编号从 0 开始）。下方对应的 `0` 表示本机的 0 号卡；
4. `Name` 表示显卡的名称，一般为显卡的型号。下方对应的 `NVIDIA GeForce RTX 4060 ...` 表示 0 卡是 `RTX 4060`，后面的 `...` 是未完整显示的显卡名称；
5. （不重要）`Driver-Model` 表示驱动运行的模式。常见模式：（1）`Windows Display Driver Model` (WDDM)[1]：

[1] https://en.wikipedia.org/wiki/Windows_Display_Driver_Model

面向图形和显示，GPU 同时用于图形渲染和计算；（2）`Tesla Compute Cluster` (TCC)[2]：面向计算的模式，GPU 完全用于计算；

6. `Pwr:Usage/Cap` 表示当前/最大功率。下方对应的 `1W / 80W` 表示当前功率 1 W，最大 80 W；
7. `Memory-Usage` 表示显存 (Video RAM, VRAM) 的使用情况。下方对应的 `72MiB / 8188MiB` 表示总显存为 8188 MiB （约 8 GiB，称 0 卡是 `8G RTX 4060`），当前已用 72 MiB；
8. `GPU-Util` 表示 GPU 核心的利用率。下方对应的 `0%` 表示利用率为 0%，即空闲。

注 区分 MiB/MB 与 GiB/GB：

1. 1 MiB = 2^{20} Bytes，1 GiB = 1024 MiB；
2. 1 MB = 10^6 Bytes，1 GB = 1000 MB

下方的 `Processes` 表示使用 GPU 的进程表，其中的重要参数：

1. `GPU` 表示进程占用的 GPU 的编号；
2. `PID` 和 `Process name` 分别表示占用 GPU 的进程编号和进程名；
3. `Type` 表示进程使用 GPU 的形式：（1）`C`：Compute，计算任务；（2）`G`：Graphics，图形任务（如桌面显示、图像和视频的渲染等）；（3）`C+G`：同时有计算和图形任务；
4. `GPU Memory Usage` 表示进程占用的 VRAM。WDDM 模式下，进程级的 VRAM 占用一般显示 `N/A`，但不代表进程不占用 VRAM。这是因为 WDDM 将 VRAM 虚拟化，则驱动不会将进程级的 VRAM 占用情况暴露给 `nvidia-smi`。故 WDDM 模式下，一般只通过 GPU 概览表中的 `Memory-Usage` 或任务管理器查看总 VRAM 占用情况。

注意事项：

1. 当代（2025）主流的 CUDA 版本是 11.3 / 11.6 / 11.8 / 12.1 / 12.2+，一般驱动版本 525.xx+ 即可兼容；
2. 此处的 `CUDA Version: 12.7` 不是当前的 CUDA 版本，而是指该驱动跑不超过 12.7 版本的 CUDA 都没问题，因为 CUDA 理论上向下兼容。然而，实践中最好还匹配工程项目的驱动版本和 CUDA 版本，尤其是 CUDA 版本，否则有时会出现奇怪的问题。当然，不代表匹配了就不会出现奇怪的问题。另一方面，`CUDA Version: 12.7` 有时也能强行跑 12.7+ 版本的 CUDA，但不保证不会出现奇怪的问题。总之，大部分情况下 CUDA 版本与项目使用的 CUDA 版本匹配即可；
3. 某些多年前经典论文的官方实现可能采用较低版本的 CUDA，有时用更高版本的 CUDA 会出现奇怪的问题。解决方法：（1）设置多个版本的 CUDA，请自行查阅教程（不保证不会出现奇怪的问题）；（2）经典的论文一般都有很多民间实现和优化，在 GitHub 上查找年份较新的民间的仓库，一般可兼容当代（2025）主流的 CUDA 版本；
4. 口语化的“*把显卡跑满*”有两层意思：（1）在*不显存超限*（俗称“*爆显存*”，即 CUDA out of Memory，简称 CooM）的前提下，VRAM 占用尽量接近最大 VRAM。一般通过调大 `batch_size` 实现；（2）让 GPU 利用率尽量接近 100%。GPU 利用率低常是因为*数据加载慢、并行度低、任务太小*。解决：① 批处理/数据并行，如将多次小计算合并为一次大计算；② 异步并行：加载数据与 GPU 计算同时进行，避免 GPU 等待 CPU 加载数据；③ 避免频繁在 CPU 与 GPU 间拷贝数据；④ 提高 `batch_size`，充分发挥 GPU 能力；⑤ 混合精度训练，加速训练，并减小 VRAM 占用，进而可进一步提高 `batch_size`；⑥ 算子融合：将多个运算融合为一个算子。

1.1.1.3 `nvcc -V`

NVIDIA CUDA Compiler (nvcc) 用于编译 `.cu` 文件。在终端运行

```
nvcc -V
```

输出

```
(no_pain_diffusion) PS C:\Users\Hytidel> nvcc -V
```

[2] https://docs.nvidia.com/gameworks/content/developertools/desktop/tesla_compute_cluster.htm

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Tue_Mar__8_18:36:24_Pacific_Standard_Time_2022
Cuda compilation tools, release 11.6, V11.6.124
Build cuda_11.6.r11.6/compiler.31057947_0
```

中 `Cuda compilation tools, release 11.6, V11.6.124` 表示 CUDA Toolkit 编译器版本号（俗称 *CUDA 版本*）为 11.6，小版本号（基本用不到）为 11.6.124。因 CUDA 版本 11.6 不超过驱动支持的最高版本 12.7，故理论上可正常使用。

1.1.2 配置环境

在终端运行

```
conda create -n no_pain_diffusion python=3.10 -y

conda activate no_pain_diffusion
```

创建一个 `Python 3.10`（或更高版本）的 conda 环境，并激活。

运行

```
python --version
```

输出 Python 版本，我的版本是 3.10.18。

```
(no_pain_diffusion) PS C:\Users\Hytidel> python --version
Python 3.10.18
```

安装 `torch` 2.1.x（或更高版本）和对应版本的 `torchvision`。版本可参考[3]。建议安装 CUDA 11.8 对应的 `torch` 和 `torchvision`，即下方的 URL 结尾的 `cu118`。虽然我的 CUDA 版本是 11.6，但仍能按 11.8 安装。

```
pip install torch==2.3.1 torchvision==0.18.1 --index-url https://download.pytorch.org/whl/cu118
```

检查 CUDA 是否可用[4]。

```
import torch

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

输出 `True` 即为可用。

安装 `requirements.txt` 中的其它依赖。

```
pip install -r requirements.txt
```

1.1.3 找到 conda 环境的路径

在终端运行：

```
conda env list
```

[3] https://pytorch.org/get-started/previous-versions/

[4] `tasks/ch_1/sec_1_1/subsec_1_1_2/test_cuda.py`

结果：

```
(no_pain_diffusion) PS C:\Users\Hytidel> conda env list
# conda environments:
#
base                  D:\anaconda3
d2l-gpu               E:\JupyterNotebookEnv\envs\d2l-gpu
no_pain_diffusion  *  E:\JupyterNotebookEnv\envs\no_pain_diffusion
```

其中的星号表示当前激活的 conda 环境，[5]是 conda 环境 `no_pain_diffusion` 的安装路径。

在路径后接 `/Lib/site-packages` 即为该 conda 环境安装的库的目录，如我的是[6]。

在扩散模型的科研中，常需在 `diffusers` 库的代码的基础上修改。`diffusers` 库的代码可在上述路径后接 `/diffusers` 找到。

1.2 Python 基础

1.2.1 Python 语法

当代（2025）主流的 Python 版本为 3.10+，不建议使用 3.9 及以下的 Python 版本，因为 3.10 中的某些语法已被广泛使用。

正如买玩具要看说明书，学软件要看官方文档，因为官方文档是容易被忽略却又最权威的学习资料，免费的官方文档的价值远高于大部分需要付费的课程。Python 3.10 的官方文档的根目录为[7]，在页面上方可切换语言（英文/中文）和 Python 版本。官方的教程 (tutorial) 在[8]。

虽然 Python 官方文档是全面、具体、权威，但读者学习起来可能稍显枯燥，且并非其中所有的语法都是工程实践中常用的。因此，更推荐新手看民间的**视频**教程。此处强调“视频”是因为读者能直观地看到完整的操作流程，避免纯图文教程“跳步”带来的困惑。换句话说，初学软件不建议只看书。这也是本课程除了本讲义外，还提供视频讲解的原因。

具体地，笔者推荐如下的视频教程作为新手入门教程：

1. 【莫烦 Python】Python 基础教程[9]；
2. 3 小时超快速入门 Python | 动画教学【2025 新版】【自学 Python 教程】【零基础 Python】【计算机二级 Python】【Python 期末速成】[10]。这是一套 2025 年 2 月更新的动画丰富的 Python 教程。

若想更深入地了解 Python 的某个功能（函数、库、特性等），可参考：

1. 博主码农高天（Python 核心开发者）的视频[11]；
2. 在 Python 的官方文档中搜索[12]；
3. 善用搜索引擎。

Python 语法的视频教程在当代并不稀缺。如果你花钱学 Python 语法是为了找个老师手把手教，我认为是合理的；但若你花钱是买不含任何售后、答疑的录播课，那我认为价值远低于给本课程一键三连。

[5] `E:/JupyterNotebookEnv/envs/no_pain_diffusion`

[6] `E:/JupyterNotebookEnv/envs/no_pain_diffusion/Lib/site-packages`

[7] https://docs.python.org/3.10/

[8] https://docs.python.org/3.10/tutorial/index.html

[9] https://www.bilibili.com/video/BV1wW411Y7ai

[10] https://www.bilibili.com/video/BV1Jgf6YvE8e

[11] https://space.bilibili.com/245645656/upload/video

[12] https://docs.python.org/3

1.2.2 Typing 库

虽然 Python 定义变量时无需显式地写出变量类型，而是能根据值自动推断，但在某些场景（尤其是函数传参）中标注变量类型可增加可读性，减小用户阅读文字说明的时间成本。

下面介绍常见用法，更多可参考[13]。

1.2.2.1 常用数据类型

常用变量类型的标注[14]。

常用 collection 类型的标注，对 Python 3.9+[15]；对 Python 3.8 及更早的版本（事实上这种标注方法在当代的工程项目中仍常用），需从 Typing 库 import 对应的类型名称（首字母大写）[16]。

1.2.2.2 多种数据类型

函数传参中，某个参数可能可传多种数据类型[17]。对 Python 3.10+，可用 | 连接类型，表示“或”；对 Python 3.9 及更早的版本，需从 Typing 库 import `Union` 类型。

1.2.2.3 可选参数

函数传参中，某些参数并非一定要传入[18]。对 Python 3.10+，可用 | `None` 表示可选参数；对 Python 3.9 及更早的版本，需从 Typing 库 import `Optional` 类型。

1.2.2.4 类名

在 [19]实现 `Person` 类。在目录[20]下（原因见1.2.6节）运行[21]。

无需实际使用类名（如创建对象），仅为标注数据类型时，为类名加引号可避免 import，有利于避免循环 import（见1.2.8节）。

1.2.3 `if __name__ == "__main__"`

`__name__` 用于在 Python 项目（单文件或多文件系统）中唯一地识别模块[22]：

1. 通过 `import` 导入模块时，模块的 `__name__` 是其真实名称（非别名）；
2. 特别地，“`__main__`”表示 Python 程序的*入口点* (entry point) 或*顶级代码环境* (top-level code environment)，即程序以哪个 `.py` 文件为起点开始运行。

在每个 `.py` 文件中，通过 `if __name__ == "__main__"` 判断该文件是否是程序入口点，可实现每个小模块除了被其它文件 import 外也可独立运行，方便单独调试。如，在[23]中实现加法函数 `add()` 并单独测试。在主文

[13] https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html

[14] `tasks/ch_1/sec_1_1/subsec_1_2_2/single_types.py`

[15] `tasks/ch_1/sec_1_1/subsec_1_2_2/collection_types_3_9.py`

[16] `tasks/ch_1/sec_1_1/subsec_1_2_2/collection_types_3_8.py`

[17] `tasks/ch_1/sec_1_1/subsec_1_2_2/union.py`

[18] `tasks/ch_1/sec_1_1/subsec_1_2_2/optional.py`

[19] `tasks/ch_1/sec_1_1/subsec_1_2_2/person.py`

[20] `tasks/ch_1/sec_1_1/subsec_1_2_2`

[21] `tasks/ch_1/sec_1_1/subsec_1_2_2/test_person.py`

[22] `tasks/ch_1/sec_1_2/subsec_1_2_3/test_name.py`

[23] `tasks/ch_1/sec_1_2/subsec_1_2_3/models/add_utils.py`

件[24]中 import 加法函数并使用。

在 AI 科研的实践中，该方法常用于在定义神经网络的架构的类的文件中测试网络的输出维度是否正确。如，在[25]中实现 `MyMLP` 类并单独测试。

1.2.4 Pathlib 库

不同操作系统的路径的分隔符可能不同，如 Windows 系统常为“\”(backslash)，而 UNIX 系统常为“/”(slash)。为保证程序的健壮性（或称鲁棒性，robustness），需保证代码中的路径可适配不同的操作系统。Python 中，Pathlib 库对 os 库进行封装，提供了更方便的接口。

下面介绍常见用法，更多可参考[26]。

本节的代码需在目录[27]下运行（运行本节的 `.ipynb` 默认在此目录）。

1.2.4.1 创建路径、基本路径操作

创建一个表示1.2.2节的代码的目录的路径，并进行 navigate、判断文件是否存在、判断文件类型（文件/目录、相对路径转绝对路径、路径的组成 [28]。

1.2.4.2 文件名、后缀

对路径对象，`.name` 表示文件名（含后缀），`.suffix` 表示后缀，`.stem` 表示文件名（不含后缀）[29]。

值得注意的是，若一个目录的名称为“`path_w_.dot`”，虽然它不是文件类型，但 `.dot` 仍会被视为后缀，则 `.stem` 的结果为 `path_w_`。

1.2.4.3 遍历

非递归、递归地遍历路径，找到符合要求的子路径[30]。

1.2.4.4 创建路径、删除路径

创建/删除单个文件/目录[31]。

传参涉及路径的函数（如读取、保存文件）中，传入的路径一般是 `str` 或 `Path` 类型。建议在操作路径前，保证路径是 `Path` 类型，以适应不同的操作系统。

Pathlib 库无法递归地删除非空目录，可用 Shutil 库（1.2.5节）实现该功能。

[24] `tasks/ch_1/sec_1_2/subsec_1_2_3/main.py`

[25] `tasks/ch_1/sec_1_2/subsec_1_2_3/models/my_mlp.py`

[26] https://docs.python.org/3.10/library/pathlib.html

[27] `tasks/ch_1/sec_1_1/subsec_1_2_4`

[28] `tasks/ch_1/sec_1_1/subsec_1_2_4/path_basic.py`

[29] `tasks/ch_1/sec_1_1/subsec_1_2_4/name_suffix_stem.ipynb`

[30] `tasks/ch_1/sec_1_1/subsec_1_2_4/traverse.ipynb`

[31] `tasks/ch_1/sec_1_1/subsec_1_2_4/crete_and_delete.ipynb`

1.2.5 Shutil 库

1.2.6 工作路径

1.2.6.1 展示工作路径

1.2.6.2 相对路径

1.2.6.3 切换工作路径

1.2.7 Jupyter Notebook

1.2.7.1 切换工作路径

1.2.7.2 注意事项

1.2.8 `import`

1.2.9 `locals()`、`globals()`、`SimpleNamespace`

1.2.10 断点、调试

1.2.11 `generator`、`yield`

1.2.12 文件读写

1.2.13 面向对象

1.2.13.1 Python 面向对象基础语法

1.2.13.2 `__call__()`

1.2.13.3 静态变量

1.2.13.4 类方法

1.2.13.5 抽象类、抽象方法

1.2.13.6 `MethodType`

1.2.14 `arg`、`kwargs`

1.2.15 `getattr()`

1.3 常用 Python 库

本节介绍 AI 科研中常用的部分 Python 库。可视化和绘图相关的库见2章，深度学习相关库见3章，机器学习相关库见4章，数学相关库（如数值计算等）见5章。

第二章 可视化与绘图基础

第三章 深度学习基础

第四章 机器学习基础

第五章 数学基础