Task 0

基本概念理解

1、什么是“模型”？机器学习中的模型是如何工作的？

答：模型是对现实世界中或者某个系统进行简化和抽象之后的表示形式；

机器学习模型是一种算法的表达，它通过梳理海量数据来查找模式或进行预测。机器学习 (ML) 模型由数据助力，是人工智能的数学引擎。

ML模型的简要分类

ML模型类型	用例
linear线性回归分类	财务电子表格等数字数据格式
图形模型	欺诈检测或者情绪感知
决策树/随机森林	预测结果
深度学习神经网络	计算机视觉和自然语言处理等

对于几个小概念的区分：

- **AI 模型**是一个广义概念，涵盖机器学习、深度学习模型，以及基于规则的系统、专家系统等其他技术。凡是展现出智能行为的模型，都属于此范畴。
- **机器学习模型**是 AI 模型的子集，利用统计方法从数据中学习，无需明确编程。它们可以采用多种技术，包括但不限于神经网络。
- **深度学习模型**是机器学习模型的进一步细分，依靠多层人工神经网络从数据中进行学习。它们在图片识别和语音识别等复杂任务中尤为有效。

机器学习工作流程：

- **1.获取数据**
- **2.数据基本处理**
- **3.特征工程**
- **4.机器学习(模型训练)**
- 5.模型评估

  ```
  - 结果达到要求，上线服务
  - 没有达到要求，重新上面步骤
  ```

2. 模型没有生物的意识和记忆，它们是如何“学习”的？

这种模型的学习不是生物学意义上的学习，而是一个**数学上的“优化”过程。**它更像是一个不断“试错”和“修正”的循环。

这个过程主要依赖三个关键：

1. **损失函数**：
   这是一个用来衡量模型当前“预测”与“真实答案”之间差距的函数。预测得越离谱，损失函数计算出的“损失值”（一个数字）就越大。
2. **优化器**：
   这是一个算法（最经典的是**梯度下降法**），它的作用是根据损失值，计算出应该**如何微调**模型内部的每一个参数，才能让下一次的损失值**变得稍微小一点。**
3. **大量的训练数据**：
   这是模型学习的原材料。

学习的过程如下：

1. **预测**：从训练数据中拿一个样本（比如一张猫的图片和“猫”这个标签），让模型进行一次预测。模型可能会瞎猜成“狗”。
2. **计算损失**：损失函数比较模型的预测“狗”和真实答案“猫”，发现差距很大，于是计算出一个很大的损失值。
3. **计算梯度**：优化器根据这个损失值，反向计算出模型里成千上万个参数，各自应该朝哪个方向（增大还是减小）调整一点点，才能让模型下次更可能猜成“猫”。
4. **更新参数**：优化器根据计算出的方向，对模型的所有参数进行一次微小的更新。
5. **重复**：不断地重复以上1-4步，用成千上万张不同的图片去训练。每一次重复，模型的参数都会被微调得更好一点，损失值也会变得越来越小。

最终，当损失值低到一定程度时，我们就说模型“学会了”。它的“记忆”并不是存在于大脑海马体里，而是**固化在了那上百万个经过反复优化的参数值中。**这些参数共同构成了一个能识别特定模式的数学结构。

3. 什么是监督学习？什么是无监督学习？请分别举一个例子。

这两种学习方式的核心区别在于**训练数据有没有“标准答案”。**

- **监督学习**
  - **定义和目标**：训练模型时，提供给它的数据是**带有明确标签或“标准答案”**的，最终他会学习一个从输入到输出的映射关系
  - **例子**：
    - **图像分类**：给模型成千上万张图片，每张图片都明确标注了是“猫”还是“狗”。模型学习后，就能对一张新的、没有标签的图片进行分类。
    - **房价预测**：给模型很多房子的信息（面积、地段、房龄），并告诉它每套房子的真实成交价。模型学习后，就能根据一套新房子的信息，预测出它的价格。
- **无监督学习**

- **定义以及目标**：训练模型时，提供给它的数据是**完全没有标签或"标准答案"**的。让他自己去发现这些石头可以按颜色、大小、形状等方式来分类，但并不告诉他具体要分成几类或者分类的标准是什么。最终的训练目标是从数据本身发现隐藏的结构、模式或规律。
- **例子**：
  - **聚类**：给模型一大堆用户的购物数据，模型可以在不知道任何用户信息的情况下，自动把用户分成几个群体，比如"高消费年轻群体"、"爱买打折品群体"、"深夜购物群体"等。
  - **降维**：当数据特征太多时，模型可以自动学习哪些特征是冗余的，哪些是核心的，从而用更少的特征来表示原始数据，便于分析和可视化。

4. AI 是什么？深度学习和传统机器学习的区别?

- **AI 是什么?**
  人工智能 是一个非常宽泛的、总体的概念。它的终极目标是创造出能够像人类一样思考、推理、学习和解决问题的智能机器或系统。它是一个宏大的领域，而不是某一项具体的技术。

他们之间的关系就像是一个套娃**AI > 机器学习 > 深度学习**

- **机器学习** 是实现 AI 的一种**重要方法。**它不是让机器遵循人类编写的硬性规则，而是让机器**从数据中自动学习**出规律和模式。
- **深度学习** 则是机器学习领域中一个**非常强大的分支。**
- **深度学习 vs. 传统机器学习的核心区别：**

  核心区别在于**"特征工程 "**这一步是否需要人工干预。

  1. **传统机器学习**：
     - **依赖人工特征工程。**在把数据喂给模型之前，需要由人类专家或工程师，凭借经验和领域知识，**手动地从原始数据中提取出有用的特征。**模型学习的是这些被人工处理过的特征之间的关系。比如说你需要先告诉模型什么是"车轮"，什么是"车窗"，什么是"车门"，把这些特征提取出来（比如计算图像中圆形的数量、方形的数量），然后再让模型（如SVM、决策树）去学习"有4个车轮和若干车窗的就是汽车"。
  2. **深度学习** ：
     - **自动完成特征学习（端到端学习）。**这不需要告诉模型什么是车轮或车窗。你直接把**原始的、未处理的数据**喂给一个非常深的神经网络。比如一个深度神经网络，它的浅层网络会自动学习识别一些基础特征（如边缘、颜色块），中层网络会把基础特征组合成复杂一些的特征（如轮胎、车灯），高层网络再把这些组合成更高级的概念（如车的外形），最终判断出这是一辆汽车。整个特征提取的过程是模型在训练中**自动学到**的。

  其他区别：

  - **数据量**：深度学习通常需要比传统机器学习大得多的数据量才能表现出色。
  - **计算资源**：深度学习模型庞大，训练和推理都需要大量的计算资源（尤其是GPU）。
  - **可解释性**：传统机器学习模型（如决策树）通常更容易解释，而深度学习模型像一个"黑盒子"，很难说清它做出某个决策的具体原因。

编程与开发环境

- 我安装的是**Miniconda**

- 起初我在用命令行安装，由于网络问题下载失败，最终改用手动下载安装到windows再利用WSL2和Windows文件系统共享机制移动到ubuntu下
- ```
  yys@LAPTOP-K7DORK8R:~/JotangRecrument-MlSys-Yinyueshi$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
  --2025-08-01 17:32:18--  https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
  Connecting to 10.255.255.254:7890... failed: Connection refused.
  ```
- ```
  yys@LAPTOP-K7DORK8R:~/JotangRecrument-MlSys-Yinyueshi$ cp /mnt/c/Users/yys/Downloads/Miniconda3-latest-Linux-x86_64.sh ~/
  yys@LAPTOP-K7DORK8R:~/JotangRecrument-MlSys-Yinyueshi$ ls
  ML  MLSys  README.md
  yys@LAPTOP-K7DORK8R:~/JotangRecrument-MlSys-Yinyueshi$ ls ~
  Anaconda3-5.2.0-Linux-x86_64.sh  JotangRecrument-MlSys-Yinyueshi  Miniconda3-latest-Linux-x86_64.sh  anaconda3
  yys@LAPTOP-K7DORK8R:~/JotangRecrument-MlSys-Yinyueshi$ cd ~
  yys@LAPTOP-K7DORK8R:~$ bash Miniconda3-latest-Linux-x86_64.sh
  ```
- **安装成功**

  ```
  ==> For changes to take effect, close and re-open your current shell. <==

  Thank you for installing Miniconda3!
  ```
- **配置py3.9开发环境，首先需要手动确认使用协议**
- ```
  (base) yys@LAPTOP-K7DORK8R:~/JotangRecrument-MlSys-Yinyueshi$ conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/main
  accepted Terms of Service for https://repo.anaconda.com/pkgs/main
  (base) yys@LAPTOP-K7DORK8R:~/JotangRecrument-MlSys-Yinyueshi$ conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/r
  accepted Terms of Service for https://repo.anaconda.com/pkgs/r
  (base) yys@LAPTOP-K7DORK8R:~/JotangRecrument-MlSys-Yinyueshi$ conda create -n mlsys-task python=3.9 -y
  2 channel Terms of Service accepted
  ```
- **激活开发环境**
- ```
  (base) yys@LAPTOP-K7DORK8R:~/JotangRecrument-MlSys-Yinyueshi$ conda activate mlsys-task
  ```

- **由于硬件限制，没有英伟达的显卡，所以安装不了CUDA**
- **安装工具包**

```
(mlsys-task) yys@LAPTOP-K7DORK8R:~/JotangRecrument-MlSys-Yinyueshi$ conda install numpy matplotlib scikit-learn -y
2 channel Terms of Service accepted
Channels:
 - defaults
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done
```

- **安装pytorch，cpu版本的**

```
(mlsys-task) yys@LAPTOP-K7DORK8R:~/JotangRecrument-MlSys-Yinyueshi$ conda install pytorch torchvision torchaudio cpuonly -c pytorch
2 channel Terms of Service accepted
Channels:
 - pytorch
 - defaults
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done
```

- **最后两个简答题**
  - **问题一：为什么需要 python 虚拟环境？在命令行中如何在不使用 conda 指令的情况下使用指定虚拟环境？**
    - **第一部分的答案：** 是为了实现**项目隔离**和**环境可复现**，从而彻底解决不同项目间的“依赖冲突”问题。

随着我们参与的项目增多，会遇到一个典型困境：

- **项目 A** 是一个维护中的老项目，它依赖于一个稳定但老旧的库，比如 LibraryX v1.0。
- **项目 B** 是一个需要应用最新技术的新项目，它需要 LibraryX v2.0 版本才能使用其新功能。

如果没有虚拟环境，这两个项目将共享同一个全局的 Python 环境。无论我们安装哪个版本的 LibraryX，都会导致另一个项目因版本不兼容而崩溃。这种情况被称为“依赖地狱”。

虚拟环境通过创建独立的、互不干扰的“沙箱”来解决此问题：

1. **依赖隔离:** 我们可以为项目 A 创建一个名为 env_A 的虚拟环境，并在其中只安装 LibraryX v1.0。同样，为项目 B 创建 env_B，并在其中安装 LibraryX v2.0。这两个环境就像两个独立的工具箱，里面的工具版本互不影响，保证了各自项目的稳定性。
2. **环境可复现:** 我们可以将某个虚拟环境中所有包及其精确版本号导出为一个清单文件（如 requirements.txt）。团队中的其他成员或未来的自己，可以用这个文件一键创建出完全相同的开发环境，确保代码在任何地方都能以同样的方式运行，极大地提升了协作效率和项目的可靠性。

- **第二部分的答案：** 通过**直接调用该虚拟环境下可执行文件的绝对路径。**

  详细阐述：

  conda activate <环境名> 这个命令的本质，是修改当前终端的 PATH 环境变量，将指定虚拟环境的 bin 目录（存放着 python, pip 等可执行文件）临时性地添加到 PATH 的最前端。这样，当你在终端输入 python 时，系统会优先找到并使用这个虚拟环境里的 Python 解释器。

  如果我们不使用 activate，就等于放弃了这个自动的“快捷方式”。但我们依然可以直接“点名”要用哪个 Python 解释器。

  一个用 Conda 创建的名为 mlsys-task 的虚拟环境，其 Python 解释器的标准绝对路径通常位于：~/miniconda3/envs/mlsys-task/bin/python

  因此，我们可以通过以下命令来精确地使用它执行脚本：

```
# 直接指定解释器的完整路径来运行脚本
~/miniconda3/envs/mlsys-task/bin/python  your_script.py
```

  Use code with caution.Bash

  这个方法绕过了对 PATH 环境变量的修改，以一种更明确、更底层的方式直接调用了目标环境的程序，保证了命令的精确执行。

- **问题二：在你配环境的过程中，哪些东西需要编译？哪些不需要？**

  答案： 在本次环境配置流程中，我们安装的几乎所有软件都**不需要我们手动编译**，因为我们使用的是**预编译的二进制包** 。只有当需要从源代码安装时，才涉及编译过程。

  第一部分：不需要编译的

  无论是安装 Miniconda 本身，还是通过 conda install 安装 Python、PyTorch、Numpy 等库，我们利用的都是 Conda 强大的包管理能力。

  Conda 会根据你的操作系统（Linux x86_64）和指令，直接从云端仓库中为你下载一个**已经被官方提前编译、打包好的成品**（通常是 .tar.bz2 格式的压缩包）。这个包里包含了可以直接运行的二进制文件和所需的库。本地所做的工作主要是**解压缩和建立文件链接**，而非从头开始的编译。

  因此，以下操作均不涉及手动编译：

- 安装 Miniconda。
- conda create 创建新环境和安装指定版本的 Python。
- conda install 安装 PyTorch, Numpy, Matplotlib 等所有科学计算包。

这种方式的优点是**快速、便捷且极少出错。**

第二部分：需要编译的

当一个软件没有提供针对你系统的预编译包，或者你需要一个高度定制化的、甚至是开发中的版本时，就需要**从源代码 (Source Code) 进行编译。**

我们需要自己去下载一堆.cpp, .h 等源代码文件，然后亲自使用如 gcc, g++, cmake, make 等编译器和构建系统，在电脑上把这些组装成最终可执行的二进制文件。

在本次任务中，虽然我们没有手动执行，但以下情况属于编译范畴：

- 我们通过 git submodule 下载的 cutlass 仓库，其本身就是一套用于 GPU 计算的 C++ 源代码模板库。如果我们要在项目中真正地利用它来生成一个定制化的内核，这个过程就涉及编译。
- 如果我们想修复 PyTorch 的一个 bug，就需要下载它的全部源代码，修改后在本地完整地重新编译整个 PyTorch 库来进行测试。