

八斗学院精品班@NLP 与大模型 课程笔记

课程环境

- **Anaconda 安装:** [env-anaconda 安装](#)
 - 作用: 用于创建深度学习环境, 以及对 python 编程的支持
- **Pycharm 安装:** [env-pycharm 安装](#)
 - 作用: 用于编写 python 代码, 运行课程中的代码
- 基础 Python 环境: <https://share.note.youdao.com/ynoteshare/index.html?id=e71cc06fcbfed64c29b738613e36fe86>
- 额外的 Python 库:
 - fastapi[standard]
 - 作用: 现代、快速 (高性能) 的 Python Web 框架, 用于构建 API
 - 命令: `pip install fastapi[standard] -i https://pypi.tuna.tsinghua.edu.cn/simple`
 - mypy
 - 作用: 静态类型检查工具, 它使用 Python 的类型提示 (Type Hints) 来检查代码中的类型错误。
 - 命令: `pip install mypy -i https://pypi.tuna.tsinghua.edu.cn/simple`

目标岗位 (大模型应用工程师 / 自然语言处理算法工程师)

工作职责

- **模型应用开发:** 负责设计、开发和部署基于大模型的应用程序和工具, 如智能问答系统、智能内容生成、智能客服等, 以提升产品的智能化水平和用户体验。
- **模型优化与调优:** 对大模型进行微调 (Fine-tuning)、提示工程 (Prompt Engineering)、强化学习人类反馈 (RLHF) 等操作, 以提高模型在特定任务或领域的性能。
- **数据处理与管理:** 清洗、标注和构建高质量的训练数据集, 优化数据管道, 确保数据的有效性和可用性。
- **技术研究与创新:** 跟进大模型领域的前沿技术动态, 探索新的应用场景和解决方案, 如多模态大模型的应用、模型的高效训练方法等。

- **跨部门协作：**与产品、运营、业务等团队紧密合作，理解业务需求，将技术实现与业务场景紧密结合，推动项目的进展。

技能要求

- **专业知识：**熟悉大模型的底层原理、主流训练和部署框架，了解自然语言处理（NLP）、深度学习等领域的基础知识。
- **编程能力：**熟练掌握 Python 等编程语言，以及 PyTorch 等深度学习框架，能够高效地实现算法和模型。
- **数据处理技能：**具备数据挖掘、清洗、增强的能力，能够处理大规模非结构化数据。
- **模型训练经验：**有大模型训练（如 SFT、PEFT、RLHF 等）经验者优先，能够根据业务需求设计合理的训练方案。
- **跨学科能力：**了解不同领域的知识，如计算机视觉（CV）、推荐系统等，以便将大模型应用于更广泛的场景。

后端开发工程师-大模型应用/25k-50k

字节跳动 [查看该公司所有职位 ▶](#) 收藏 投简历

职位描述：

职位职责：

- 1、负责各个行业大模型智能体应用的开发与探索；
- 2、推动产品架构和核心技术改进和优化，解决工程技术难题；
- 3、关注大模型前沿技术，跟进业内最新研究进展和应用趋势，提出创新思路和方向，并落地。

职位要求：

- 1、具备熟练的Python/Go语言编程能力和良好的编程习惯和代码管理能力；
- 2、对语言大模型、多模态大模型、智能代理Agent技术以及大模型的部署流程有基本的了解和认识；
- 3、能够独立负责模块级别的服务开发，完成复杂组件的技术设计和实现；
- 4、熟练掌握常用的数据库，缓存，消息队列等，了解具体使用场景和性能区别、优缺点；
- 5、热爱技术，主动负责，乐于直面挑战；能够保持开放、持续学习，善于发现问题，具备良好的团队合作精神。

加分项：

- 1、熟悉大模型应用算法能力优先，包括但不限于SFT、RAG、Agent、PE等；
- 2、有大模型相关项目或使用经验者优先，熟悉Langchain、Dify等框架；
- 3、熟悉云原生相关开发使用经验者优先。

AI Agent开发工程师/20k-40k·15薪

经纬恒润 [查看该公司所有职位 ▶](#)

 收藏

立即沟通

职位描述：

一、岗位描述

- 1、深度挖掘AI产品在业务上的场景需求，推动AI能力在业务场景的持续创新落地；
- 2、负责相关Agent开发，并推动LLM、RAG等技术与工程系统的深度集成，形成汽车行业Agent设计和实现；
- 3、持续跟进LLM领域的最新技术趋势，结合AI产品需求，提供创新的解决方案，并推动技术的迭代和升级。

二、任职要求

- 1、本科及以上学历，计算机、人工智能、数学、自动化等相关专业；
- 2、熟练掌握Python编程语言，具有至少3年以上后端开发经验，能独立完成项目技术开发；
- 3、有大模型训练、微调或者Agent开发项目经验优先考虑；
- 4、优秀的分析和解决问题的能力，对解决具有挑战性的问题充满激情，具备良好的沟通和团队合作能力。

面试内容：基础知识（计算机基础、数据结构算法）

课程笔记

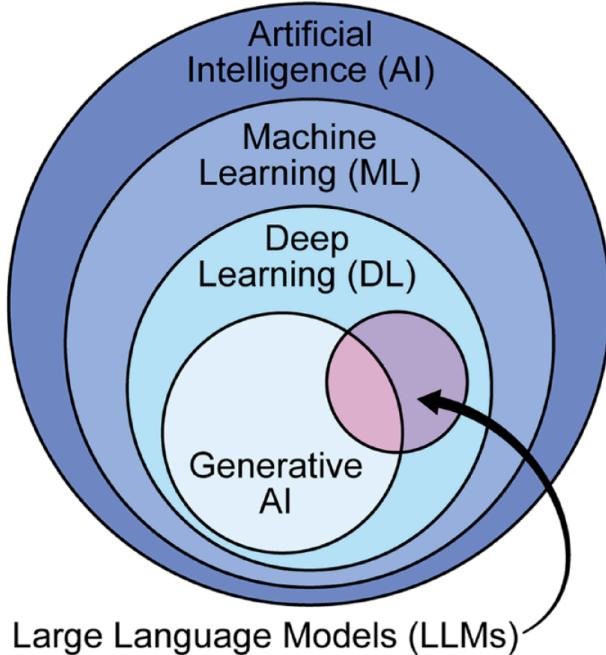
第一周：课程介绍与大模型基础

- 人工智能 (AI)、机器学习、深度学习

人工智能 (Artificial Intelligence, AI) 的目标是让机器能够像人类一样思考、学习和解决问题。你可以把 AI 理解为一个宏大的愿景或一个宽泛的领域，它涵盖了所有让机器展现“智能”的技术。机器学习 (Machine Learning, ML) 是实现人工智能的一种主要方法。它的核心思想是：与其明确地告诉机器该怎么做，不如让机器自己从数据中学习规律和模式。**深度学习 (Deep Learning, DL)** 是机器学习的一个子集。它借鉴了人脑神经网络的结构，构建了**深度神经网络 (Deep Neural Networks)**。这里的“深度”指的是网络中的层次非常多。

SQL

- 1 人工智能 (AI) > 机器学习 (ML) > 深度学习 (DL)
- 2 人工智能 是最广泛的概念，是让机器拥有智能的总称。
- 3 机器学习 是实现人工智能的一种具体方法，让机器通过数据进行学习。
- 4 深度学习 是机器学习的一种特殊类型，它使用多层神经网络，在许多复杂任务中表现非常强大。
- 5
- 6 深度学习是机器学习的一种，而机器学习是人工智能的一种实现方式。



- 自然语言处理 约等于 大模型

自然语言处理 (Natural Language Processing, NLP) 是一门让计算机能够理解、解释、处理和生成人类语言（例如我们日常使用的中文、英文等）的学科。它的核心目标是弥合人类语言和计算机语言之间的鸿沟。

NLP 的常见应用：

- **机器翻译：** 谷歌翻译、百度翻译等。
- **情感分析：** 自动分析社交媒体上的评论，判断用户对某个产品或话题是正面还是负面情绪。
- **智能客服：** 聊天机器人能够理解你的问题并给出回答。
- **垃圾邮件过滤：** 自动识别邮件中的垃圾信息。
- **语音助手：** Siri、小爱同学等，将你的语音转换成文字并理解你的指令。

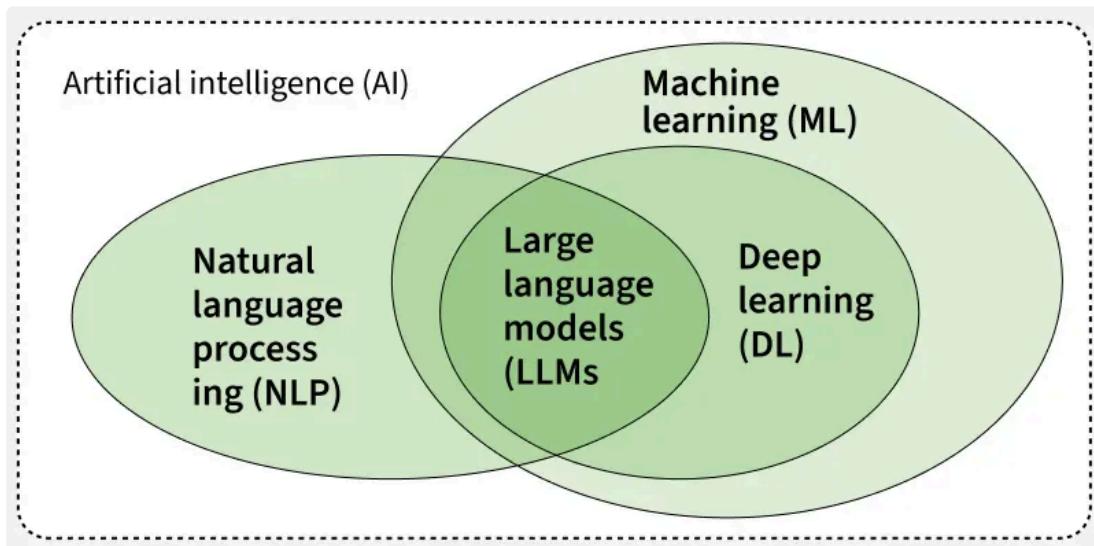
大模型 (Large Language Model, LLM) 是近年来在 NLP 领域取得突破性进展的一类**深度学习模型**。它们之所以被称为“大”，是因为拥有巨大的参数量和庞大的训练数据。

- **通用性强：** 过去你需要为每个任务训练一个模型，而现在一个大模型经过训练后，可以通过简单的指令（被称为“Prompt”）完成多种多样的任务，比如写文章、编程、回答问题、摘要、翻译等，甚至可以进行创意性的生成。这种能力被称为“**涌现能力**”。
- **强大的生成能力：** 大模型不仅能理解语言，还能流畅、连贯地生成语言。像我们熟悉的 ChatGPT、Gemini 等，它们都是大模型的典型代表。

- **上下文理解能力：** 大模型在处理长文本时，能够记住前面的信息，从而进行有逻辑、有上下文的对话。

SQL

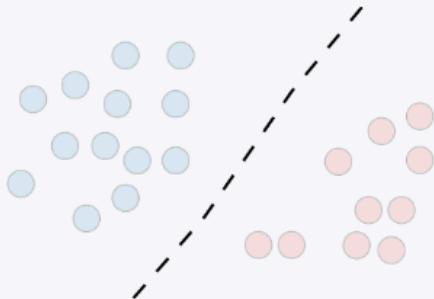
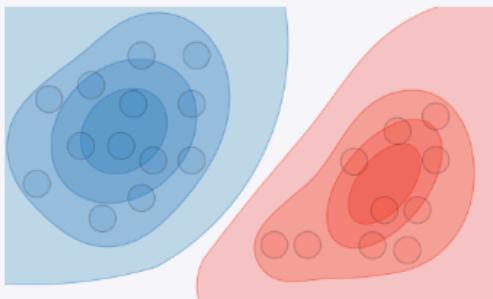
- 1 大模型是自然语言处理领域的一个重大技术突破和主流趋势。
- 2 自然语言处理（NLP）是一个广阔的领域，包含了许多理论和方法，大模型是其中最成功、最前沿的一种。
- 3 在大模型出现之前，NLP 任务通常采用传统的机器学习或深度学习方法，这些方法往往是针对特定任务设计的。
- 4 而大模型的出现，改变了 NLP 的研究范式。它不再是“为每个任务单独设计模型”，而是“用一个通用的大模型来完成几乎所有任务”。



- 判别模型、生成模型

判别模型 的核心任务是：根据给定的输入，直接学习如何划分不同类别的边界。

生成模型 的核心任务是：学习数据的内在分布和生成数据的过程。

	Discriminative model	Generative model
Goal	Directly estimate $P(y x)$	Estimate $P(x y)$ to then deduce $P(y x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes

特点	判别模型 (Discriminative Model)	生成模型 (Generative Model)
任务目标	找到不同类别之间的决策边界。	学习数据的内在分布。
主要功能	判别 / 分类。	生成新数据，也可以用于分类
模型类比	“分类专家”，只关心如何区分。	“模仿大师”，学习数据的生
典型应用	图像分类、文本情感分析等。	图像生成、文本生成、语音合

- Python2、Python3

Python 2 和 Python 3 是两种不同版本的 Python 编程语言。Python 2 在处理多国语言（如中文）时，编码问题很麻烦，而 Python 3 从一开始就以更好的方式解决了这个问题。Python 2 早已停止官方支持，不再进行安全更新和 bug 修复。目前绝大多数的现代库、框架和项目都只支持 Python 3。

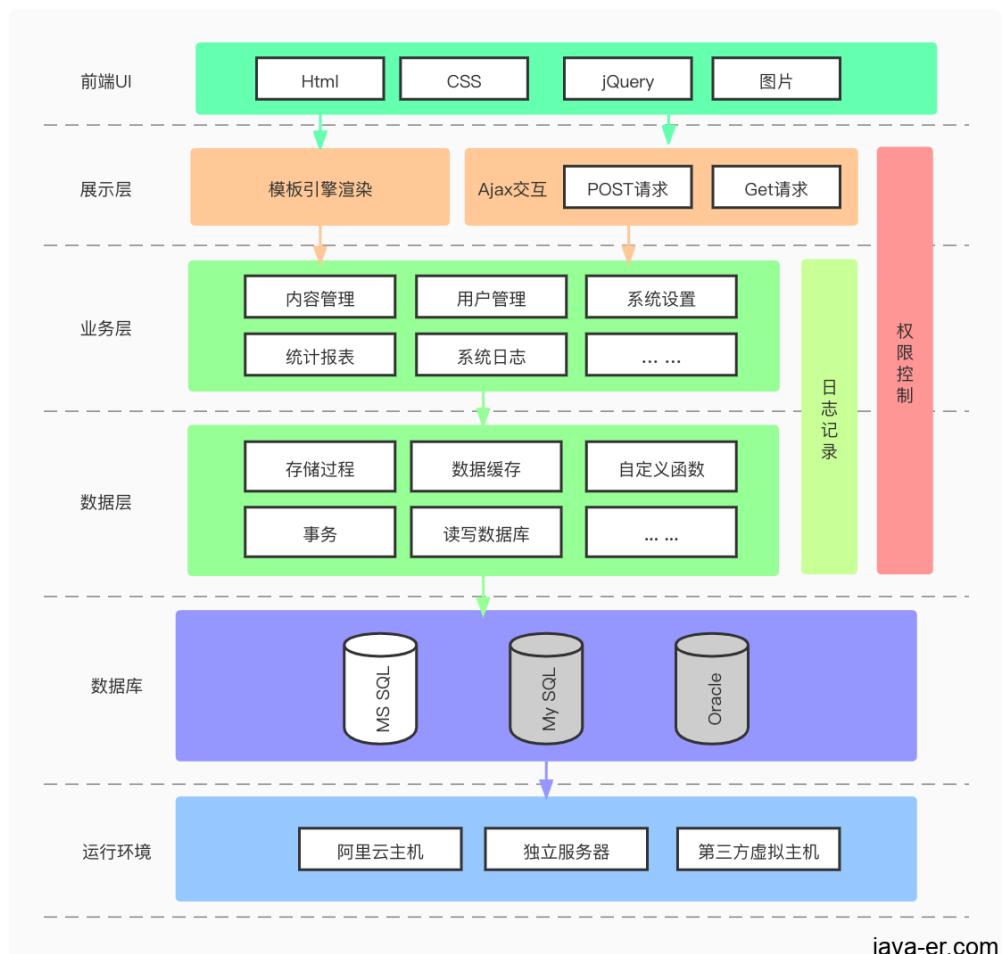
- 类型注解

类型注解 (Type Hinting)，也叫类型提示，是一种在 Python 3.5+ 版本中引入的语法，允许开发者在变量、函数参数和函数返回值后面，用冒号 `:` 来标明它们的类型。类型注解能立刻让你知道每个变量或参数的预期类型是什么，避免不必要的猜测，让代码意图更清晰。`MyPy`、`Pyright` 等第三方静态分析工具可以利用类型注解，在不运行代码的情况下，检查出潜在的类型错误。这就像一个“语法检查器”，能在你运行代码前就发现问题。

Python

```
1 name: str = "Alice"
2 age: int = 30
3 is_active: bool = True
4
5 def greet(name: str) -> str:
6     return f"Hello, {name}"
7
8 def calculate_sum(numbers: list[int]) -> int:
9     return sum(numbers)
```

- 后端、前端、算法



前端 负责与用户直接交互的部分，也就是你在浏览器、手机应用或电脑软件上看到

后端 负责处理所有你看不到的逻辑和数据。它就像一座建筑的“地基”、“承重结构”

算法 是一系列解决问题的清晰指令或步骤。它就像一个“大脑”或者“食谱”，指导

和点击的一切。它就像一座建筑的“外墙装修”和“室内设计”，目标是让用户界面美观、易用且响应迅速。

前端开发需要掌握的核心技术通常包括：

- **HTML (HyperText Markup Language):** 定义网页的结构和内容，比如标题、段落、图片等。
- **CSS (Cascading Style Sheets):** 控制网页的样式，比如颜色、字体、布局等。
- **JavaScript (JS):** 实现网页的交互功能，比如点击按钮后弹出窗口、动画效果、数据加载等。

和“水电系统”，支撑着整个应用的运行。

后端开发的核心工作包括：

- **处理数据和业务逻辑：** 比如用户登录、提交订单、发布帖子等操作，都是由后端处理的。
- **管理数据库：** 存储和管理所有用户、商品、文章等数据。
- **服务器：** 确保应用能稳定、高效地运行，并响应前端的请求。

计算机如何高效地完成任务。算法并不是一个独立的职位，而是所有工程师都需要掌握的核心技能。

算法工程师 则是专门研究、设计和优化算法的专家。他们的工作通常集中在：

- **解决复杂问题：** 比如如何让搜索引擎更快地返回结果，如何推荐用户可能感兴趣的的商品。
- **数据分析和机器学习：** 设计算法来从大量数据中提取有价值的信息，构建智能系统。
- **性能优化：** 确保代码运行得更快，占用更少的内存。

- Linux 服务器

Linux 服务器 就是运行 **Linux 操作系统** 的计算机，专门用来提供各种网络服务。Linux 提供了丰富的命令行工具，让管理员可以远程高效地管理服务器，进行自动化配置和脚本编程。几乎所有的主流编程语言、数据库和 Web 服务器软件（如 Apache、Nginx）都有 Linux 版本，并且拥有庞大的社区支持。几乎所有的云计算平台（如亚马逊 AWS、谷歌云、阿里云）都以 Linux 作为其核心操作系统。

- 企业内部员工的组成

角色名称	职责
产品经理 (肯定有)	指导团队生产符合预期的人工智能模型并成功部署
数据科学家 (总监、项目经理, 肯定有)	分析用户需求和场景, 找出数据驱动的解决办法, 型。
数据工程师	提供高质量数据, 供模型训练和线上服务使用。
机器学习 / 深度学习 / 大模型 工程师 (目标岗位)	在真实环境训练模型, 部署到线上系统提供服务。
DevOps 工程师	完成集成模型预测服务到用户应用程序, 记录行为
运维工程师 (大公司肯定有、小公司研发就是运维)	保证人工智能线上系统的稳定运行。
软件开发工程师 (肯定有, 软件工程师, 后端开发)	写业务代码, java、go

第二周：大模型使用与深度学习基础

- 信息检索、检索模型、全文检索、反向索引

信息检索 (Information Retrieval, IR) 是从海量非结构化数据中 (比如文本、图片、音频等), 找到与用户需求相关的信息。最常见的应用就是搜索引擎, 你输入一个查询词, 系统返回一堆网页链接。

检索模型 (Retrieval Model) 是信息检索背后的“大脑”, 它决定了如何评估一个文档与查询词之间的相关性, 并对搜索结果进行排序。我们可以把检索模型看作一个打分系统, 它会给每个文档打一个分数, 分数越高, 代表文档越相关, 也就越靠前显示。

全文检索 (Full-Text Search) 是一种对文档中的所有文本内容进行索引和搜索的技术。它不像传统数据库那样只搜索特定的字段, 而是对文档的全部内容进行分析, 建立索引, 从而实现更精准和更全面的搜索。

反向索引 (Inverted Index) 是实现全文检索的核心数据结构。我们可以把它想象成一本书的词汇表。在这个词汇表中, 每个词都指向所有包含这个词的页码 (或文档)。当你搜索一个词时,

系统直接在反向索引中找到这个词，然后就能迅速找到所有相关的文档，而不需要遍历每一个文档。

- TFIDF、BM25

TF-IDF (Term Frequency-Inverse Document Frequency) 是一个经典的统计方法，用来评估一个词对于一个文档的重要性。它由两部分组成：

- **TF** (Term Frequency)：词频。一个词在文档中出现的次数。出现次数越多，通常说明这个词越重要。
- **IDF** (Inverse Document Frequency)：逆文档频率。一个词在所有文档中出现的频率。如果一个词在很多文档中都出现（比如“的”、“是”），那么它对区分文档的价值就不大。IDF 的作用就是降低这些常见词的权重。

BM25 (Best Matching 25) 是一种更先进的排序算法，它是 TF-IDF 的改进版。BM25 在计算词语的重要性时考虑了更多因素，比如文档的长度、词频的饱和度等，因此在实际应用中通常比 TF-IDF 效果更好。

- 提示词工程

提示词工程 (Prompt Engineering) 是一门“与大语言模型对话”的艺术和科学。通过精心设计和调整给模型的**提示词** (Prompt)，我们可以引导模型生成更准确、更相关、更有帮助的回答。这就像给一个聪明但需要指引的助手下达指令，指令越清晰，结果就越好。

- RAG

RAG (Retrieval-Augmented Generation) 是一种结合了检索和生成的技术。它的工作流程是：

- 检索**：当用户提出问题时，首先从一个私有的知识库或数据库中**检索**出相关的文档或信息。
- 增强**：将检索到的信息作为**增强**内容，连同用户的原始问题一起，作为**提示词**输入给大语言模型。
- 生成**：模型基于这些**增强**信息来**生成**回答。

- 神经元、激活函数、全连接层

神经元 (Neuron) 是神经网络中最基本的计算单元，它模仿了人脑中的神经细胞。一个神经元接收多个输入信号，每个信号都有一个**权重**，这些输入信号加权求和后，再通过**激活函数**处理，最终产生一个输出。

激活函数 (Activation Function) 决定了神经元的输出。它的主要作用是引入**非线性**，让神经网络能够学习和处理更复杂的模式和关系。常见的激活函数有 ReLU、Sigmoid 等。如果没有激活函数，无论神经网络有多少层，它都只能执行线性运算，能力会大大受限。

全连接层 (Fully Connected Layer) 或**全连接网络** (Fully Connected Network) 是指神经网络中的一层，其中每个神经元都与前一层的所有神经元相连接。这种连接方式使得每个神经元都能接收到前一层的所有信息，并对其进行综合处理。

- 梯度下降

梯度下降 (Gradient Descent) 是训练神经网络最常用的**优化算法**之一。你可以把它想象成一个人在山坡上**下山**的过程：他每走一步，都会朝着坡度最陡峭的方向往下走，直到走到山谷的最低点（即损失函数达到最小值）。

“坡度最陡峭的方向”就是**梯度**，而“下山”就是不断调整网络参数（权重和偏置）的过程，以减少模型预测与实际值之间的误差（**损失**）。

- 深度学习优化器

深度学习优化器 (Deep Learning Optimizer) 是一类更高级的梯度下降算法，它们旨在提高训练效率和效果。常见的优化器包括：

- **Adam**: 目前最流行和最有效的优化器之一，它结合了多种优点，能更快地找到最优解。
- **RMSprop**: 能有效地处理非平稳目标。
- **SGD** (Stochastic Gradient Descent): 最基础的梯度下降变体，每次只用一个样本或一小批样本来计算梯度，速度快但可能不稳定。

课程作业

第一周

- 1、配置好 python 环境，包括常见的 jieba、sklearn、pytorch 等
- 2、使用 dataset.csv 数据集完成文本分类操作，需要尝试 2 种不同的模型。

第二周

- 1、调整 09_深度学习文本分类.py 代码中模型的层数和节点个数，对比模型的 loss 变化。
- 2、调整 06_torch 线性回归.py 构建一个 sin 函数，然后通过多层网络拟合 sin 函数，并进行可视化。

课堂提问

- 目标岗位需要会很多语言嘛，只会 Python 可以不？java 用处大吗

作为**大模型应用开发工程师**，主要工作是基于大模型进行应用开发和调优，因此 **Python 是核心且必备的语言**。

目前，绝大多数的大模型框架（如 PyTorch、TensorFlow）和主流的工具库都是基于 Python 开发的，并且社区支持最完善。因此，**如果你只掌握 Python，也完全可以胜任这个岗位**。

Java 语言在一些特定的企业级应用、大数据处理（如 Hadoop、Spark 生态）和后端开发中依然有广泛应用。对于大模型应用开发而言，如果你的团队或项目需要将大模型服务集成到基于 Java 的后端系统中，那么了解 Java 会有帮助，但这通常不是首要技能要求。

- 那大模型基本做的都是后端吗？会涉及到前端吗？额外学习后端语言会锦上添花吗？

为什么大模型应用开发偏向后端？

- 模型部署和推理：大模型的运行需要强大的算力和大量的内存，这通常都在服务器端完成，由后端负责。
- API 开发：为了让前端应用、移动应用或其他服务能够调用大模型的能力，后端需要开发和维护稳定的 API 接口。
- 数据处理和管理：大模型应用往往需要处理、清洗和管理海量数据，例如检索增强生成（RAG）应用就需要后端来管理向量数据库和文档处理流程。

通常会有专门的**前端工程师**来负责这些 UI 界面的开发，而你作为后端工程师则提供底层的 API 支持。不过，在一些小型团队或者追求“全栈”能力的岗位上，你可能会需要编写一些前端代码，但这不是核心职责。

- 数据库是指 oracle 这些吗？中间件这块课程会教嘛？

Oracle 确实是一个非常著名的数据库软件，它属于**关系型数据库**的一种。这类数据库以表格的形式存储数据，数据之间通过主键和外键建立关系，非常适合需要严格数据结构和事务一致性的场景。

除了 Oracle 这样的关系型数据库，大模型应用还会用到很多其他类型的数据库（文档数据库、向量数据库）。这些数据库通常是为了处理非结构化数据和支持大模型特有的需求而设计的。

- 感觉有些大模型都要把前端代替了？那是不是有些岗位也是全栈？

大模型确实正在改变前端的开发模式，甚至在某些方面替代了部分传统前端工作，比如：

- 文本到界面的生成（Text-to-UI）：直接输入“生成一个有登录按钮和注册表单的页面”，大模型就能帮你写出 HTML、CSS 和 JavaScript 代码。
- 自动化 UI 组件生成：基于用户需求，模型可以自动创建和修改 UI 组件，大幅提高开发效率。

大模型虽然强大，但它无法完全替代人类的创意、审美和对用户体验的深刻理解。前端工程师的核心价值将从“写代码”更多地转向**“产品设计”**和**“用户体验（UX）”**。你的工作重点会是：

- **Prompt 工程与模型调优：**更好地与大模型沟通，让它生成你想要的、高质量的前端代码。
 - **整合与优化：**将大模型生成的代码和组件整合到复杂的项目中，并进行性能优化。
 - **创新人机交互：**设计和开发基于大模型的新型交互方式，比如语音控制、多模态交互等。
-
- agent 我业务中接触的就是大模型 + 业务流程组成的。

你需要将大模型的能力封装成易于调用的接口（API），并确保它们能无缝地嵌入到业务流程中。这可能涉及到：

- **提示工程（Prompt Engineering）：**精心设计给大模型的指令（Prompt），让它更好地理解业务需求并给出高质量的回答。
- **模型微调（Fine-tuning）：**如果通用大模型在你的特定业务领域表现不佳，你可以用业务数据对模型进行微调，使其更懂你的业务。
- **性能优化：**确保大模型的响应速度足够快，能满足业务流程的实时性要求。

- 需要英语好吗？招聘对数据结构相关的算法要求高吗

绝大多数大模型和机器学习领域的最新技术、框架文档（如 PyTorch、TensorFlow、Hugging Face）以及顶尖学术论文（如 ArXiv 上的）都是以英文发布的。如果你能熟练阅读英文技术资料，就能第一时间掌握行业前沿动态，这会让你在技术上保持领先。

招聘方更看重你如何将数据结构与算法应用到实际问题中。例如，当你需要处理大量文本数据时，如何用哈希表或字典树来提高搜索效率？当你需要管理知识库数据时，如何选择合适的数据库（如上文提到的向量数据库）和数据结构来支持高效的检索？

工作重心将更多地集中在**自然语言处理（NLP）**和深度学习相关的算法上，而不是 LeetCode 上那些经典的算法题。你需要了解**大模型底层原理、神经网络结构、提示工程**以及各种**微调技术**的算法思想。

- 关系数据库、向量数据库区别；图形数据库会有讲解吗，知识图谱方向的关系型数据库是传统的主流数据库，如 Oracle、MySQL、SQL Server。
 - **存储方式：**以二维表格的形式存储数据，数据行和列清晰分明。
 - **数据模型：**数据之间通过主键（Primary Key）和外键（Foreign Key）建立关系，确保数据的完整性和一致性。
 - **适用场景：**适合需要严格数据结构、复杂查询和事务处理的业务，如电商订单、银行账户信息等。
 - **核心特点：**强调**数据的结构化和一致性**。

向量数据库是为处理非结构化数据而生的新型数据库，如 Milvus、Pinecone、Weaviate。

- **存储方式：**存储的是向量，也就是由数字组成的列表。这些向量是机器学习模型（特别是大语言模型）将文本、图片、音频等数据转换而来的数值表示。
- **数据模型：**数据之间没有预设的关系，而是通过**向量之间的相似度**进行匹配。
- **适用场景：**主要用于处理大规模非结构化数据的相似性搜索，如智能问答、图像搜索、推荐系统等。
- **核心特点：**强调**高效的相似性搜索**。

图数据库是专门处理以图（Graph）结构存储的数据的数据库，如 Neo4j、ArangoDB。数据以**节点（Nodes）和边（Edges）**的形式组织，节点代表实体（如“人”、“公司”），边代表实体间的关系（如“工作于”、“是朋友”）。

- agent 是一个物理上存在的机器人吗？

Agent 的核心思想是，它能感知环境、进行思考、规划行动，并执行任务。它是一个能够自主完成一系列复杂任务的智能实体。

现在，大语言模型（LLM）成为了 Agent 的“大脑”。一个基于大模型的 Agent 能够根据指令，自主地分解复杂任务，并调用各种工具来完成。

- 特征向量是什么样的，人类可读吗？

特征向量（Feature Vector）本质上就是一组用来描述事物的数字，通常以列表或数组的形式呈现。在机器学习和自然语言处理领域，特征向量的维度（即数字的数量）通常非常高，可以达到几百甚至上千。而且这些数字的含义不像上面的苹果例子那样简单直接。

- 这些机器学习法，是不是市场上有第三方工具将一些业务逻辑提取方式中实现？还是说这些特征提取等需要后台开发对于具体项目由开发者实现？

市场上有大量的第三方工具、API 和服务，它们已经把这些复杂的机器学习方法和特征提取逻辑封装好了。如果你的业务有非常独特的特征，第三方通用模型可能无法很好地捕捉。这时，你需要自己用业务数据对模型进行微调（Fine-tuning），甚至从头开始训练。

- NER 不用大模型可以用表格填充吧？

命名实体识别（NER）并不一定需要使用大模型。在许多场景下，你完全可以使用基于传统规则和机器学习的方法，甚至通过表格填充这种更简单直接的方式来实现。

- 那进到公司，多数是不是做微调的跟应用落地的。工程化能力这块？

大模型本身是一个通用的基础模型，它像一个知识渊博但缺乏专业领域的学生。直接使用通用大模型去解决公司的具体问题，效果往往不理想。。你既要懂模型，知道如何微调和优化，又要懂工程，能将模型部署并集成到实际业务中。

- 刚才提到机器学习和深度学习，要掌握到什么程度，只需要把提供的资料学会就行吗？

你需要理解机器学习中的有监督学习、无监督学习等基本概念。知道什么是模型训练、过拟合、欠拟合，以及如何评估模型的性能（比如准确率、召回率、F1 分数）。

对于深度学习，你需要了解神经网络的基本结构，比如前向传播和反向传播是如何工作的。掌握这些原理能帮助你理解为什么大模型会有好的表现，也能在模型出现问题时进行调试。

你需要熟练使用主流的深度学习框架，尤其是 **PyTorch**。掌握如何用 PyTorch 定义模型、加载数据、进行训练和推理。

- 如何基于业务场景判断使用深度学习技术还是大语言模型技术的，基于老师的经验这个边界大概在哪里？

大语言模型（LLMs）的优势在于它们的**通用知识和强大的泛化能力**。它们在海量数据上预训练，拥有丰富的世界知识和语言理解能力。当你的业务场景需要**理解和生成复杂的、开放域的、或需要依赖通用知识的文本**时，优先考虑使用大模型。

传统深度学习模型，如特定的 CNN、RNN 或 BERT 等，通常是在**特定任务和特定数据集**上进行训练的。它们更像是一个“专才”。当你的业务场景是**封闭的、结构化的、任务单一且不需要通用知识**时，传统深度学习模型往往是更高效和经济的选择。

- 如果为公司构建了内部大模型，之后只是做维护？我司 AI 岗就是推公司主流 AI 航道工具的使用，基本无开发。

大模型不是一劳永逸的。随着时间的推移，业务数据和需求会不断变化。此外还需要建立完善的监控系统，持续追踪模型的性能（如回答准确率、响应速度等）。如果发现性能下降，需要进行排查和优化。

- cursor 和 trae 那种大厂的 IDE 的这种应用涉及哪些技术？

这些 IDE 大多是基于 Web 技术构建的，比如使用 **Electron** 或 **Tauri** 框架，将前端技术（HTML、CSS、JavaScript/TypeScript）打包成桌面应用。这使得它们能够快速迭代和跨平台部署。

这些 IDE 会对基础大模型进行微调，使其更擅长生成代码。此外，它们还会利用 **RAG（检索增强生成）** 技术，在你的代码库中检索相关代码片段和文档，作为上下文提供给大模型，从而生成更符合项目风格的代码。

- 模型做微调成本好像巨大，实践价值没有那么高？需要很深入学习吗？是不是学习工作流编排和服务封装方向会更好一些？

传统的全量微调（Full Fine-tuning）成本巨大。它需要海量的算力、存储和时间，对公司来说是一笔不小的开支。对于大多数企业而言，直接用这种方式来微调大模型，实践价值确实不高。

现在有了 PEFT (Parameter-Efficient Fine-Tuning) 技术，PEFT 技术通过只训练模型中极小一部分参数，就能达到接近全量微调的效果。这大大降低了对显存和算力的要求，让微调的成本变得可控。

- 多模态, 大模型, 大语言模型, nlp 之间的关系?

NLP 是一个广阔的学科领域，是人工智能（AI）的一个分支。它的目标是让计算机能够理解、分析、生成和处理人类的语言。

大模型（Foundation Models） 是一个更广义的概念。它指的是在海量数据上预训练、具有通用能力的模型。大语言模型就是**大模型**的一种。

大语言模型（LLM） 是 NLP 领域的一项革命性技术。它们是使用深度学习技术，在海量文本数据上进行预训练而成的巨大模型。

多模态是一种技术理念，它指的是让模型能够同时处理**多种类型的数据**（即“模态”），比如文本、图像、音频、视频等。

- 大模型正常的工作状态是怎么样的，也是 996 吗？

在**互联网大厂**中，大模型团队的工作强度通常很高。为了保持技术领先，项目迭代速度快，加班（例如 996 或大小周）是比较常见的。你的工作可能非常专注于某一小块技术，但深度要求很高。

在**AI 初创公司**中，工作节奏会更快，压力也更大。由于团队规模小，你可能需要身兼数职，做更多全栈性质的工作。加班是常态，但你的成长速度也会非常快。

如果你在大模型团队，但身处**传统行业或国企**，工作强度通常会比较稳定，加班相对较少。这类公司可能更注重将大模型技术应用于内部业务优化，比如提升办公效率或改善客服流程。你的工作更多是应用落地和维护，而非高强度的技术研发。

- 在 FastAPI 架构图中看到，Python ORM 框架开发，一般用 SQLAlchemy 来实现？

SQLAlchemy 不仅支持基本的 CRUD（创建、读取、更新、删除）操作，还提供了复杂的查询构建、事务管理和连接池等高级功能。这使得它能够处理几乎所有类型的数据库操作，满足企业级应用的需求。

在 FastAPI 应用中，使用 SQLAlchemy 来处理数据库交互是一种非常常见的模式。

- **FastAPI 负责 API 和业务逻辑：** 它处理 HTTP 请求、路由、数据验证和业务逻辑。
- **SQLAlchemy 负责数据库操作：** 它处理所有与数据库相关的任务，比如连接数据库、执行查询和管理数据对象。

- 每次运行相同的代码，验证正确率，得到的结果是一样的吗？这个随机性有什么导致的呢？

在训练过程中，数据通常是打乱（shuffle）后分批次（batches）送入模型的。即使你设置了相同的随机种子，但如果数据加载器的工作线程（workers）数量不同，或者并行处理的顺序略有差异，那么每个批次的数据内容也可能不同，从而影响模型的训练过程。

- 这个 random_state 具体是什么含义？对数据集划分的具体影响是啥呀？

random_state 是一个用于控制随机过程的参数，它是一个整数。简单来说，它的作用是**固定随机过程的种子**，以确保每次运行代码时，产生的随机结果都是一样的。

- TensorFlow 和 pytorch 有什么区别啊 各种深度学习框架 有必要了解吗 老师

PyTorch 的代码风格更符合 Python 的直观习惯，它使用**动态计算图**。这意味着你在写代码时就像在编写普通的 Python 程序，可以更容易地调试和修改。这种灵活性让 PyTorch 在学术研究和快速原型开发中非常流行。

如果你刚刚入门或希望跟上最新的研究动态，**强烈建议你优先学习和精通 PyTorch**。目前，绝大多数开源大模型和最前沿的微调技术都是基于 PyTorch 实现的。

- 机器学习调参，是调整那块呀？

机器学习调参，也称为**超参数调整（Hyperparameter Tuning）**，主要是调整那些**在模型训练之前设置的参数**。这些参数不是通过模型训练自动学到的，而是由我们手动配置的。

- 真实生成环境中是使用 vscode 多一些还是使用 pycharm 多一些

PyCharm 是一款功能完备的集成开发环境 (IDE)。它提供了强大的 Python 专用功能，比如代码补全、调试、重构和测试，这些功能不需要额外配置。如果你主要专注于 **Python 后端** 和 **模型应用**，并且项目规模较大、需要处理复杂的业务逻辑，那么 PyCharm 可能会提供更高效、更集成的开发体验。

- sklearn 对应机器学习模型的应用，pytorch 是对应深度学习模型的应用，pytorch 会更重要一些吗？

Scikit-learn 的优点是**简单、高效、易于上手**。对于大多数不需要处理复杂非结构化数据（如图像、文本、音频）的任务来说，Scikit-learn 是一个非常好的选择。在学术界和研究领域，PyTorch 已经成为首选。大多数最新的大模型研究论文、新的微调方法（如 LORA）和开源模型库（如 Hugging Face Transformers）都优先支持 PyTorch。

- 老师怎么选择用机器学习模型还是深度学习模型

最适合处理**结构化数据**，比如表格（CSV、Excel 文件）中的数据。如果你的数据量不是特别大（例如，几万到几十万条），并且特征已经清晰地定义，那么传统的机器学习模型通常是更好的选择。

最适合处理**非结构化数据**，比如图像、文本、音频和视频。深度学习模型可以直接从原始数据中学习和提取特征，而无需你手动进行复杂的特征工程。此外，深度学习模型通常在**大规模数据集**上表现更佳，数据越多，其性能提升越显著。

- 机器学习，模型初始化时只是获取了一个模型实例并设置了相关参数，然后用 fit 方法计算特征向量，最后再做预测是吗？

模型初始化只是创建了一个模型的实例，并设置了它的超参数，比如决策树的深度、随机森林中树的数量等。此时，模型还没有学习任何东西，它的内部参数（例如，决策树的分支条件或线性模型的权重）都还没有确定。

在将数据交给模型之前，你需要进行**特征工程**。这是机器学习中一个非常重要的步骤，它指的是**将原始数据转换成模型能理解的特征向量**。`fit` 方法是模型“学习”的过程。在这个阶段，模型会根据你提供的**特征向量** (`x`) 和**目标标签** (`y`)，通过**优化算法**来调整其内部参数，使其能够找到特征和标签之间的映射关系。

- `n_neighbors` 参数是在预测阶段发挥作用，使用训练集中与 测试集中的数据 距离最近的 `n_neighbors` 个数据进行预测？

在 k-近邻 (K-Nearest Neighbors, KNN) 算法中，`n_neighbors` 参数决定了在进行预测时，模型需要考虑距离最近的多少个训练样本。

- 老师，一般工作中，是多人合作，还是一人完成一个模型的设计？

一般一个人就可以完成模型应用的设计、开发和部署。会将模型的能力封装成 API，并集成到公司的产品或服务中。还需要进行**模型微调和提示工程**，以提升模型在特定任务上的表现。

- 机器学习的方法 KNN, 决策树等有什么区别，什么场景使用哪种？

KNN 是一种惰性学习 (Lazy Learning) 算法，因为它在训练阶段几乎不做任何工作。它只是简单地将所有训练数据存储起来。当需要对新数据进行预测时，KNN 会计算它与所有训练数据之间的距离，然后找出离它最近的 k 个邻居。最终的预测结果由这 k 个邻居的类别（分类任务）或平均值（回归任务）决定。

- fastapi 返回的都是 json 格式吗？

JSON (JavaScript Object Notation) 因为其轻量、易于读写和跨语言兼容的特性，已经成为 Web API 中最主流的数据交换格式。

- **HTML 页面**: 你可以使用 `fastapi.responses.HTMLResponse` 来返回一个完整的 HTML 页面。
- **文件流**: 你可以使用 `fastapi.responses.FileResponse` 来返回文件，比如图片、PDF 或其他二进制文件。
- **纯文本**: 你可以使用 `fastapi.responses.PlainTextResponse` 来返回纯文本内容。

- 通过 fastapi 开发后端应用有没有类似 mvc 的架构呢？

FastAPI 本身没有强制采用 MVC (模型 - 视图 - 控制器) 架构，但在实际开发中，开发者们通常会借鉴 MVC 或更现代的分层架构 (Layered Architecture) 思想来组织项目。

- **用户**发送 HTTP 请求到 **Router**。
- **Router** 调用 **Service** 层的方法。
- **Service** 层调用 **Database Access** 或其他服务来处理业务逻辑。
- **Database Access** 层与数据库交互，返回数据。
- **Service** 层处理数据，并返回给 **Router**。
- **Router** 根据 **Pydantic Model** 定义的格式返回 JSON 响应给用户。

- FastAPI 和 Django 是找大模型相关工作都需要的技能还是 fastAPI 多一点？

大模型推理通常是计算密集型任务，对响应速度有很高要求。FastAPI 基于 **异步编程** 设计，这使得它能够同时处理大量并发请求，而不会因为一个请求在等待模型推理结果而阻塞整个服务。这对于高并发场景下的 API 服务至关重要。相比之下，Django 传统上是同步的，虽然新版本也增加了异步支持，但在这方面，FastAPI 具有原生优势。

- 在 Python 和 Java 后端进行交互时，选择使用 FastAPI 还是 gRPC 框架？

场景	选择 FastAPI (RESTful API)	选择 gRPC
通用性和易用性	你的服务需要与多种类型的客户端（前端、移动应用）交互，并且团队更熟悉 HTTP 协议和 JSON。	你的服务只在后端微服务之间调用，并且需要严格的类型校验和高性能。
性能要求	中低性能要求，或者你的服务瓶颈不在网络传输，而是在计算密集型任务（如大模型推理）。	极高的性能要求，尤其是在处理高并发、低延迟的内部服务调用时。
数据传输	传输的数据量较小，或者不需要频繁的流式传输。	需要传输大量数据或实现实时流式传输（如语音识别、实时监控）。

- jieba 分词是啥原理？

jieba 分词自带一个**大规模的词典**，包含了常用的词语及其词频信息。这些词频是根据大规模语料库统计出来的，反映了词语在中文中出现的频率。

jieba 会使用**动态规划算法**，结合词典中每个词的词频，计算出每条路径的**总概率**。总概率最高的路径，就是被认为是最合理的分词结果。

- 请问 sklearns 是将中文翻译成英文再进行的分词吗？这样会不会导致歧义？

Sklearn 主要用于处理**数值型数据**。当你想用它来处理文本数据时，你需要先进行**特征工程**，将文本转换成数值向量。这个转换过程通常是使用其他的专门工具来完成，而不是 sklearn。

- **分词**：首先，使用像 **jieba** 这样的中文分词库，将中文句子切分成一个个独立的词语。
- **停用词过滤**：移除像“的”、“是”、“了”这些没有实际意义的词。
- **文本向量化**：使用像 sklearn 提供的 **CountVectorizer** 或 **TfidfVectorizer** 工具，将分词后的词语转换成数值型的特征向量。
- **请注意**：这里的 sklearn 只是作为**工具**来完成向量化，它并没有进行分词或翻译。
- **模型训练**：最后，将这些数值向量输入给 sklearn 的机器学习模型进行训练。

- 如果是某个垂直领域的专业术语很多，jieba 分词都没有的话怎么做词典？

如果一个垂直领域（比如医疗、金融、法律）有大量专业术语，而这些词语不在 jieba 的默认词典中，那么分词结果就会很糟糕，这直接影响后续的模型效果。

- CountVectorizer 和 transformer 转向量的区别是什么？

CountVectorizer 是一个非常经典的文本向量化方法，它属于词袋模型（Bag-of-Words）的一种。Transformer 是一种强大的深度学习模型，它通过大规模的预训练来学习词语的语义和上下文关系。

- 新词发现算法生成的数据集是不是还得微调，如果数据集很大的话人工调整的工作量是不是也很大

新词发现算法通常是基于统计学原理（如词频、凝固度、自由度等）来工作的。它能找到那些在语料库中频繁出现，且内部结构稳定的词语。

- 大模型开发更多的是处理数据，或者清洗数据，平时工作的工作量会很大吗？

数据决定了模型的上限，而算法只是在逼近这个上限。

- 工作时需要自己构建数据集吗？

你需要自己构建数据集，但这通常是团队合作的结果，并且不是从零开始。根据业务需求，对数据进行人工或半自动标注。例如，如果你在做智能问答系统，你需要标注出每个问题对应的正确答案。如果你在做情感分析，你需要标注出每条评论的情感倾向。

- 我们后面做的项目，会涉及到海量数据处理吗？有这样的数据集可以使用吗？

在实际工作中，你处理的“海量数据”通常来自公司内部的各种系统，比如用户日志、客服对话记录、产品文档等。

- 微调大模型指的是改开源模型的源码吗？后面会讲大模型的部署方法吗？

微调大模型通常不是修改开源模型的源码，而是基于开源模型的权重和架构，通过少量数据进行参数更新。大模型的部署方法是课程的重点内容，也是大模型应用开发工程师的核心技能。

课程资料

机器学习

机器学习是一门人工智能的子学科，其核心思想是让计算机系统通过**数据 (data)** 学习，而不是通过明确的编程指令来完成特定任务。简单来说，就是让机器自己从数据中找到规律和模式，并利用这些规律来做出预测或决策。

机器学习方法通常可以分为三大类：

- **监督学习 (Supervised Learning)**：这是最常见的类型。在这种学习中，模型使用带有**标签 (labeled)** 的数据进行训练。每个数据点都有一个对应的正确答案（标签）。模型的目标是学习从输入到输出的映射关系。
 - **分类 (Classification)**：预测离散的类别，比如将邮件分为“垃圾邮件”或“非垃圾邮件”。
 - **回归 (Regression)**：预测连续的数值，比如预测房价、股票价格或气温。
- **无监督学习 (Unsupervised Learning)**：在这种学习中，模型使用**没有标签 (unlabeled)** 的数据进行训练。模型的目标是发现数据中隐藏的结构和模式。
 - **聚类 (Clustering)**：将相似的数据点分组到一起，比如将客户按行为模式进行细分。
 - **降维 (Dimensionality Reduction)**：减少数据的特征数量，同时保留最重要的信息，比如主成分分析 (PCA)。
- **强化学习 (Reinforcement Learning)**：模型通过与环境进行交互来学习。它会根据**奖励 (rewards)** 和**惩罚 (penalties)** 来调整自己的行为，以达到最终的目标。比如，一个AI下棋程序通过赢棋获得奖励，输棋获得惩罚，从而不断提升自己的棋艺。

机器学习的常见模型类型

- **线性模型 (Linear Models)**：这类模型通过找到一个最优的**线性方程 (linear equation)** 来描述输入特征和输出之间的关系。
 - **线性回归 (Linear Regression)**：用于回归任务，试图找到一条最佳拟合直线（或超平面）。
 - **逻辑回归 (Logistic Regression)**：尽管名字里有“回归”，但它实际上是用于二分类任务的，通过一个**S型函数 (sigmoid function)** 将输出映射到 0 到 1 之间，表示概率。
- **决策树 (Decision Trees)**：这类模型通过一系列**if-then-else** 的规则来做出决策，形成一个树状结构。每个内部节点代表一个特征上的测试，每个分支代表一个测试结果，而叶节点则代表最终的类别或数值。
- **k-近邻算法 (k-Nearest Neighbors, KNN)**：这是一种基于实例的算法。对于一个新的数据点，它会找到训练集中与它最相似（距离最近）的 **k** 个数据点，然后根据这 **k** 个点的标签来决定新数据点的标签。例如，如果是分类任务，就以**多数投票 (majority vote)** 的方式决定类别。

数据划分 (Data Splitting)

在机器学习中，我们通常不会用所有数据来训练模型，而是将其划分为不同的子集，以评估模型的性能。最常见的划分方式是：

- **训练集 (Training Set)**: 用于训练模型的数据，模型通过学习这些数据中的模式来建立其内部参数。
- **验证集 (Validation Set)**: 用于在训练过程中调优 (tune) 模型超参数 (hyperparameters) 的数据。例如，我们可以用验证集来决定决策树的深度，或者神经网络的层数。
- **测试集 (Test Set)**: 用于最终评估模型性能的数据。这部分数据在训练和调优过程中是完全不可见 (completely unseen) 的，确保我们对模型在新数据上的表现有一个公正的评估。

深度学习

神经网络 (Neural Networks)

- **神经元 (Neuron)**: 这是神经网络最基本的组成单元，它接收输入信号，通过一个激活函数 (Activation Function) 进行处理，然后产生一个输出信号。
- **层 (Layer)**: 由多个神经元组成，通常分为：
 - **输入层 (Input Layer)**: 接收原始数据。
 - **隐藏层 (Hidden Layer)**: 在输入层和输出层之间，用于进行复杂的计算和特征提取，深度学习的“深度”就体现在隐藏层的数量上。
 - **输出层 (Output Layer)**: 产生最终的预测结果。

训练过程中的核心要素

- **权重和偏置 (Weights and Biases)**: 这些是模型在训练过程中需要学习的参数。**权重 (Weights)** 决定了输入信号的重要性，而**偏置 (Biases)** 则用于调整神经元的激活阈值，它们共同决定了模型的预测能力。
- **前向传播 (Forward Propagation)**: 数据从输入层通过隐藏层，一层一层地向前流动，最终到达输出层，得到一个预测结果。
- **损失函数 (Loss Function)**: 也叫**成本函数 (Cost Function)**，用于衡量模型的预测结果与真实值之间的差距。损失值越小，代表模型性能越好。
- **反向传播 (Backpropagation)**: 这是训练神经网络的核心算法。它利用**梯度下降 (Gradient Descent)** 的原理，从输出层反向计算损失值对每个权重和偏置的梯度，然后根

据梯度来更新 (update) 这些参数，以最小化损失函数。

- **优化器 (Optimizer)**: 负责在反向传播过程中，根据梯度来更新模型参数。常见的优化器包括 Adam 和 SGD (Stochastic Gradient Descent)。

其他概念

- **激活函数 (Activation Function)**: 给神经网络引入非线性 (non-linearity)。如果没有激活函数，无论网络有多少层，都只能学习到线性关系。常见的激活函数有 ReLU、Sigmoid 和 Softmax。
 - **ReLU (Rectified Linear Unit)**: 最常用的激活函数，计算简单且能有效解决梯度消失 (Vanishing Gradient) 问题。
 - **Softmax**: 常用于多分类任务的输出层，它能将输出转换为概率分布，所有输出值的总和为 1。
- **过拟合和欠拟合 (Overfitting and Underfitting)**
 - **过拟合 (Overfitting)**: 模型在训练集上表现非常好，但在新的、未见过的数据上表现很差。这通常是因为模型学习了训练数据中的噪声和偶然特征。
 - **欠拟合 (Underfitting)**: 模型在训练集和新数据上都表现不佳。这通常是因为模型太简单，无法捕捉到数据中的基本模式。
- **正则化 (Regularization)**: 用于防止过拟合的技术，通过在损失函数中增加一个惩罚项来限制模型的复杂度，比如 L1 和 L2 正则化。
- **批次大小 (Batch Size)**: 在训练过程中，每次更新模型参数时所使用的样本数量。
- **学习率 (Learning Rate)**: 在每次参数更新时，调整参数的步长。学习率过高可能导致模型无法收敛，过低则会使训练过程非常缓慢。

Pytorch

核心数据结构：张量 (Tensors)

- **张量 (Tensor)** 是 PyTorch 中最基本的数据结构，它类似于 NumPy 的 ndarray，但支持在 GPU 上运行。
- **创建张量**: 可以通过多种方式创建张量，例如从 Python 列表或 NumPy 数组转换，或者使用 `torch.rand()`、`torch.zeros()` 等函数直接创建。

Plain Text

```
1 import torch
2 import numpy as np
3
4 # 从 Python 列表创建
5 x = torch.tensor([[1, 2], [3, 4]])
6
7 # 从 NumPy 数组创建
8 y_np = np.array([[5, 6], [7, 8]])
9 y = torch.from_numpy(y_np)
10
11 # 创建随机张量
12 z = torch.rand(2, 3)
```

- **GPU 支持：**张量可以轻松地在 CPU 和 GPU 之间移动，从而利用 GPU 的并行计算能力来加速训练。

Plain Text

```
1 # 检查是否有可用的 GPU
2 if torch.cuda.is_available():
3     # 将张量移动到 GPU
4     x_gpu = x.to("cuda")
5     print(x_gpu)
```

自动求导：Autograd

- **自动求导 (Autograd)** 是 PyTorch 的核心功能之一，它能够自动计算所有可微分张量的梯度，这对于反向传播算法至关重要。
- **.requires_grad**：通过将张量的 `requires_grad` 属性设置为 `True`，可以告诉 PyTorch 追踪其上的所有操作，以便进行梯度计算。

Plain Text

```
1 x = torch.tensor([1.0], requires_grad=True)
2 y = x * 2
3 # y 现在也有 requires_grad=True
4 print(y.requires_grad) # Output: True
```

- `.backward()`：调用张量上的 `.backward()` 方法，可以自动计算所有叶子张量（即由用户直接创建的张量，而不是通过运算生成的）的梯度。计算出的梯度存储在 `.grad` 属性中。

Plain Text

```
1 x = torch.tensor([2.0], requires_grad=True)
2 z = x ** 2 + 3
3 z.backward() # 计算 z 对 x 的梯度
4 print(x.grad) # Output: tensor([4.])
```

- 梯度清零：在每次训练迭代中，需要清零 (**zero out**) 梯度，以防止梯度累积。

Plain Text

```
1 # 假设 model 是一个神经网络
2 # model.zero_grad() 或 optimizer.zero_grad()
```

构建神经网络：`torch.nn` 模块

- `torch.nn` 提供了构建神经网络所需的各种模块和层，例如全连接层、卷积层等。
- `nn.Module`：所有神经网络模块都继承自 `nn.Module`。
- 网络定义：一个典型的神经网络定义包括：
 - 在 `__init__` 方法中定义网络层。
 - 在 `forward` 方法中指定数据在网络中的前向传播过程。

Plain Text

```
1 import torch.nn as nn
2
3 class SimpleNet(nn.Module):
4     def __init__(self):
5         super(SimpleNet, self).__init__()
6         self.fc1 = nn.Linear(10, 5) # 全连接层，输入10，输出5
7         self.relu = nn.ReLU()
8         self.fc2 = nn.Linear(5, 1) # 输入5，输出1
9
10    def forward(self, x):
11        x = self.fc1(x)
12        x = self.relu(x)
13        x = self.fc2(x)
14
15    return x
```

- **损失函数 (Loss Function):** `nn` 模块也提供了常见的损失函数，例如 `nn.MSELoss()` (均方误差) 和 `nn.CrossEntropyLoss()` (交叉熵)。

优化器 (Optimizers)

- `torch.optim` 模块提供了各种优化算法，用于更新模型参数。
- **实例化：**优化器需要接收模型的参数 `model.parameters()` 和学习率 `lr` 等超参数。
- **优化步骤：**
 - a. 调用 `optimizer.zero_grad()` 清除旧的梯度。
 - b. 进行前向传播，计算损失。
 - c. 调用 `loss.backward()` 进行反向传播。
 - d. 调用 `optimizer.step()` 更新模型参数。

Plain Text

```
1 # 假设 model 和 loss 已经定义
2 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
3
4 # 在训练循环中...
5 optimizer.zero_grad()
6 outputs = model(inputs)
7 loss = criterion(outputs, labels)
8 loss.backward()
9 optimizer.step()
```

面试题汇总