# AI 学习时间

关键词：#引子 #书籍推荐 #大模型 #大模型简化机制

2025-03-19 (第1课) @矩阵前端研发二组

# 引子 - 实施方式

- 每周一节课，每次 20 分钟
  - 时长很短，内容精炼，希望大家都有所收获
  - 这个时长对成年人专注力要求很低，希望大家都能专心学习
- 内容根据学习大纲有序进行
  - 确保学习内容没有依赖上的矛盾
  - 确保内容的一致性与连续性
  - 确保大家可以根据大纲进行复习与预习
- 分享以不给大家增加任何负担的形式进行
  - 分享内容由 AI 学习小组组织
  - 分享内容由 AI 学习小组审核通过，确保内容质量

# 引子

## 课程特点

- 打破常规学习顺序，寻求适合团队的学习路线
- 重点讲述“是什么”以及“为什么”，“怎样做”留给大家实践
- 以通俗的方式进行讲述，同时不失严谨性与科学性

## 目标

- 让大家一同乘上 AI 浪潮的小船
- 希望通过“先会带后会”以及“共同学习”的方式提升团队的学习效率
- 提升团队整体对 AI 的认知，提升整体竞争力

# 引子 - 必要性

本来 AI 已经足够火热，理论上学习 AI 的必要性不言而喻，但是对于学习一个全新的领域的来说，强调必要性本身，也是一件很必要的事情。

## 了解概念的必要性

- **了解概念，才有尝试的方向**
  - 人的行为受概念驱使，比如当你不知道“苹果”的时候，你不可能会想着“我要去尝试吃这个东西”
  - 当你知道了“苹果”之后，你会尝试去吃，吃了之后会尝试做更多的事情，比如种苹果树，榨苹果汁等。
- **了解概念，才知道一些现象背后的原因**
  - 为什么 AI 会有幻觉，那是因为我们常说的 AI 是指“大语言模型”，而大语言模型是“概率模型”，基于概率，不是基于现实
  - 为什么 LLM 有输入长度上限，那是因为 LLM 的核心是**注意力机制**，注意力机制的核心是一个**相关矩阵**，矩阵规模的增速是输入长度的平方，受到训练资源的影响，一般在模型定义的初期，就确定了输入窗口的大小

# 引子 - 必要性

## 了解概念的必要性

- **了解概念，才有解决问题的方向**
  - 如果希望 AI 能减少幻觉，那么可能会选择**联网搜索**，或者 **RAG**
  - 如果希望 AI 能调用其他工具完成任务，那么可能会选择 **MCP**
- **了解概念，才方便交流**
  - 大家使用通用的术语进行交流，减少沟通成本
  - 在使用熟悉的术语交流时不需要进行过多解释

# 学习路线 - 传统学习路线

- **基础知识储备**
  - 数学基础：线性代数、概率统计、微积分
  - 编程基础：Python、数据结构与算法
- **核心领域**
  - 机器学习
  - 深度学习
  - 自然语言处理
  - 计算机视觉
- **实践应用**
  - 项目实战
  - 框架使用
  - 模型部署

# 学习线路 - 新的学习路线

- **概念学习**
  - 大模型
    - 大语言模型，深度学习模型，多模态
    - transformer，注意力机制
    - RAG, MCP, 微调，蒸馏
  - 深度学习
    - 求导的链式法则，反向传播，梯度下降
    - 神经网络
    - 循环神经网络（RNN, LSTM）
    - 卷积神经网络（CNN）
  - 强化学习（RL）
  - 生成对抗网络（GAN）
- **机制学习**
  - 深度学习原理
  - 大语言模型原理
  - 强化学习原理
  - 生成对抗网络原理
- **实践**
  - 穿插在前面的学习中
  - 概念学习伴随高层次的应用与实践
  - 机制学习伴随着低层次的实现

# 书籍推荐

- **《深度学习：从基础到实践》** ⭐⭐⭐⭐
  - 适合深度学习初学者的入门书，没有数学，概念解释为主
  - 涵盖深度学习的各个方面，从零开始，原理讲述，各种应用
- **《Python深度学习》** ⭐⭐⭐⭐
  - 深度学习的应用，基于 keras
  - 应用为主，原理比较简略，不过例子不错
- **《深度学习入门》** ⭐⭐⭐⭐⭐
  - 深度学习的入门，是非常易懂，生活化例子
  - 带有 python 实现
- **《深度学习进阶 - 自然语言处理》** ⭐⭐⭐⭐⭐
  - 语言模型的入门，从最基础的定义开始，到各种语言模型
  - 带有 python 实现
- **《深度学习入门2 - 自制框架》** ⭐⭐⭐⭐
  - 从零开始实现一个深度学习框架
  - 循序渐进，最后能运行，能训练

# 大模型 - 宏观定义

平时我们提到的**大模型**，一般是指**大语言模型（LLM，Large Language Model）**，但下面以更统一，更抽象的方式来描述大模型。

无论是深度学习模型，还是大语言模型，模型可以看成是一个**函数**，给它一个输入，它就给你一个输出。

```
const output = Model(input)
```

- 在 chatgpt 场景里，大家在输入框输入的，就是 `input`，而大模型返回的就是 `output`。
- 在图片生成应用里，大家在输入框输入的提示词就是 `input`，而生成出来的图片就是 `output`。

# 大模型 - 模型机制

虽然从宏观上看，模型就是一个函数，那么它内部的机制是怎样的呢？我们也可以从模块化的角度来理解。

先对输入输出进行规范，大模型的输入是一个序列（向量），无论对于文本内容，还是多媒体内容，本质上都能转化为一个序列 $\mathbf{X}$。大模型的输出也是一个序列 $\mathbf{Y}$。

比如

- 输入：“今天天气好吗”转化为序列就是 `['今', '天', '气', '好', '吗']`
- 输出：“很好”转化为序列就是 `['很', '好']`

向量可以进行任何向量空间支持的运算，而对向量进行变换，主要是通过矩阵乘法进行。一个非常基本的向量变换，可以表达为

$$\mathbf{Y} = \mathbf{W}\mathbf{X} + \mathbf{b}$$

通常会对输出进行激活函数处理，而这个**激活函数**是一个非线性函数，可以先将其看成是一个函数。所以对于模型内部的实现来说，可以简化为

```
const Model = (X) => {
    const Y = activate(W * X + b)
    return Y
}
```

# 大模型 - 层

上面得到的一个非常基础的模型抽象，我们可以将这个抽象看成是模型的一**层**。

```
const Layer = (X) => {
    const Y = activate(W * X + b)
    return Y
}
```

可以看到这个函数里面有两个**固定**参数，`W` 和 `b`，另外还有一个选定的激活函数 `activate`。为了更直观的调用，我们将其改写为一个类。

```
class Layer {
    constructor(W, b, activate) {
        this.W = W
        this.b = b
        this.activate = activate
    }
    forward(X) {
        const Y = this.activate(this.W * X + this.
        return Y
    }
}
```

通过这种方式，我们可以实例化很多层出来。

```
const layer1 = new Layer(W1, b1, activate1)
const layer2 = new Layer(W2, b2, activate2)

const Model = (X) => {
    const Y1 = layer1.forward(X)
    const Y2 = layer2.forward(Y1)
    return Y3
}
```

再把模型抽象为一个类，把这些层作为入参传入，那么我们便得到一个模型类。

```
class Model {
    constructor(layers) {
        this.layers = layers
    }
    forward(X) {
        let Y = X
        // 每一层往前计算
        for (const layer of this.layers) {
            Y = layer.forward(Y)
        }
        return Y
    }
}
```

实例化模型的时候，需要先定义好层。

```
const layer1 = new Layer(W1, b1, activate1)
const layer2 = new Layer(W2, b2, activate2)

const model = new Model([layer1, layer2])
```

# 大模型 - 参数

从上面的定义可以看到，真正决定模型输出的，其实是那些**参数**。

```
1   const W1, W2, W3, b1, b2, b3
```

所以我们常常听到的大模型参数量，主要是指这些参数的总数量。而我们使用 `ollama` 之类的工具本地部署模型的时候，实际上下载的是模型的参数以及层信息，`ollama` 给我们暴露接口调用的时候，实际上模型背后的运算就是按顺序调用 `forward` 方法。

# 大模型 - 参数

下面通过一个简单的例子感受一下参数量的规模。

比如我们希望通过使用一个模型来处理一张固定大小的图片，比如 100x100 的图片，那么我们可以定义一个模型，让它输出一张同样大小的图片。更简化一点，这是一张黑白图片，也就是只有灰度。所以图片信息实际上是一个 100x100 的序列，每个元素是一个 0-1 的数字，这个数字是灰度值。

要对一个 1w（100x100）个元素的图片进行变换，我们需要一个定义一个 1w x 1w 的矩阵 $\mathbf{W}$，也就是一亿个参数。再加上偏置 $\mathbf{b}$，一层就需要 一亿零一万 个参数了。

如果这个模型有很多层，比如十层，那这个模型就是十亿（省去零头）参数的规模。由此看出，我们现在听到的那些上百亿参数的模型，并不是虚张声势。

当然，实际应用中，需要对图片进行降维，通过层层卷积网络将其降维，通过这种技术可以将模型的参数量控制在更小的规模内。

# 小结

- **大模型**：一个将输入转换为输出的函数，本质是一系列向量变换的组合
- **层**：大模型的基本计算单元，包含权重矩阵（$\mathbf{W}$）和偏置（$\mathbf{b}$），然后通过激活函数处理输出
- **参数**：大模型的参数就是权重矩阵（$\mathbf{W}$）和偏置（$\mathbf{b}$），参数量决定了模型的规模和能力

# 实践时刻

- 使用 conda 安装于管理 AI 环境。
- 在 huggingface 上下载一些模型来玩。
- 使用 ollama + chatbox 本地部署和使用模型。
- 使用 trae 或 cursor 等工具写代码。

# Q & A