# 大模型与训练知识点

问答合集

Completed on

**Oct 18, 2025**

Prepared by

**Yuanji Zou**

## 1. 模型原理：

### Transformer

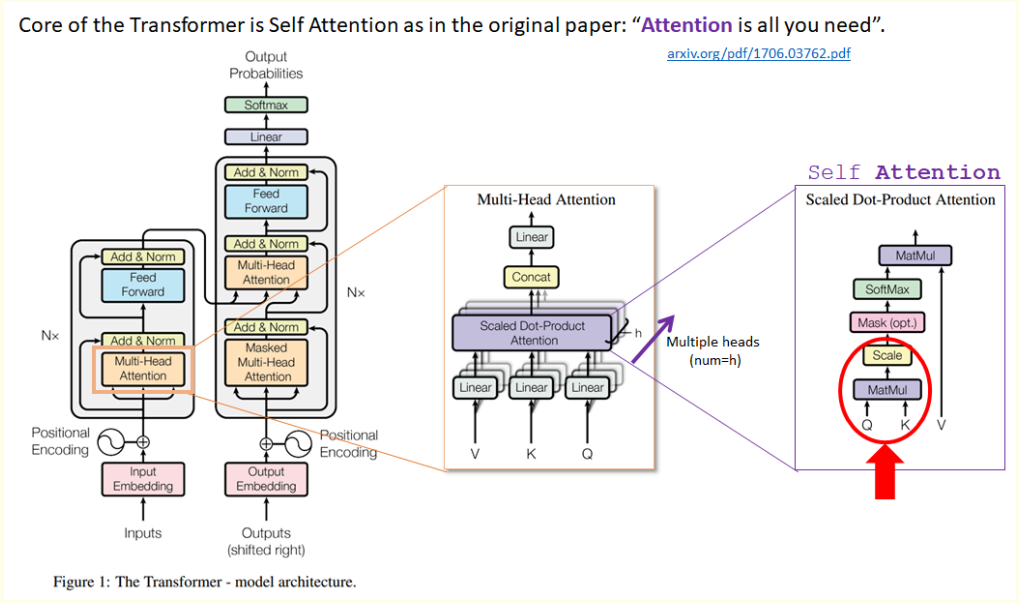
- 

Figure 1: The Transformer - model architecture.

### 架构概述：

Transformer由编码器-解码器(Encoder-Decoder)组成，主要具有以下模块：

- 编码器：
    - 多头注意力机制（Multi-Head Self-Attention）
    - 前馈神经网络（Feedforward Neural Network）
    - 归一化（Layer Normalization）
    - 残差连接（Residual Connection）
- 解码器：
    - 带掩码自注意力（Masked Multi-Head Self-attention）
    - 编码-解码自注意力（Encoder-Decoder Attention）
    - 前馈神经网络

## 编码器：

各部分作用概述：

- 位置编码 - 绝对相对位置编码，实现时序建模。将词的语义和位置信息映射到同一个向量空间，让 self-attention 可以同时处理两者，让注意力机制直接利用两者的联合相似性。同时使参数量保持不变，提高训练效率。$z_i = x_i + p_i$。$x_i$语义方向，$p_i$位置方向。
- 多头注意力机制 - Multi-Head机制通过参数拆分提升并行表征能力。
- 前馈神经网络 - $FNN(x) = W_2 Relu(W_1 x + b_1) + b_2$。位于注意力机制之后，对每个位置的表示进行独立的非线性变换。输入维度：举例：$d_{model} = 512, d_{ff} = 2048$。对于每一个token：$x \in 512 \times 1$.
  $W_1 \in 2048 \times 512, W_2 \in 512 \times 2048$。用于a) 学习非线性映射 b)对每个位置向量进行特征提取 c) 特征扩展+压缩，提升表达能力。
- 归一化
- 残差连接

## 解码器：

解码器的流程和编码器类似，但是增加了一个额外的Encoder-Decoder Attention：

1. 带掩码自注意力（Masked Multi-head Self-Attention）：逐步生成文本，因此希望满足因果约束。通过在注意力权重中屏蔽未来位置，保证每个

token 只能依赖过去的上下文，从而实现自回归（**auto-regressive**）生成。

2. Encoder-Decoder Attention：这是把 **Encoder** 编码的**“理解结果”** 和 **Decoder** 正在生成的输出 联系起来的桥梁。Query 来自 Decoder（当前生成的部分）；Key/Value 来自 Encoder（整个输入句子的语义表示）。

# 注意力机制

- 长距离依赖捕获：传统的RNN处理长序列时，存在梯度消失问题，而自注意力机制能全局捕捉所有位置的依赖关系。
- 并行化计算：由于每个token之间的计算无依赖性，Transformer能并行处理序列，大幅提高效率。
- 权重可视化。
- 处理变长输入。

数学原理（Query-Key-Value矩阵运算+Softmax归一化）：

## QKV矩阵运算：

每个输入Token通过一个线性变化映射到Query，Key，Value。

$$Q = XW^{Q},\ K = XW^{K}, V = XW^{V}$$

其中$X \in \mathbb{R}^{n \times d_{model}}$。$n$表示序列长度，$d_{model}$表示嵌入维度。

- $W^{*}, * \in \{Q, K, V\}$是三个需要学习的权重矩阵。
- $d_k$是注意力维度。通常设置为$d_k = \frac{d_{model}}{h}$。

## Self-Attention机制:

通查查询和键之间的点积计算注意力权重，并用这些权重加权值（Value）。

$$Attention(Q, K, V) = softmax(\frac{QK^{\top}}{\sqrt{d_k}})V$$

- $d_k$作为缩放因子，防止梯度消失。

### 多头注意力：

$$MultiHead(Q,K,V) = Concat(head_1,...,head_h)W^O$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

其中：

- $h$为头数，每个注意力头都有独立的权重$W_i^Q, W_i^K, W_i^V$。
- 输出连接后再投影到最终维度$d_{model}$。
- 允许模型关注输入的不同部分。提高学习能力，增强对不同特征的理解。

### 带掩码自注意力：

$$MaskedAttention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}} + M)V;$$

$$M = [M_{ij}] = 0(i \le j), else\ -\ \infty.$$

# 2. BERT，GPT，T5架构及其预训练和微调。

基于Transformer的预训练语言模型在自然语言处理领域取得了重大突破。

## 架构分类：

- Encoder模型：仅包含编码器部分，适合提取上下文表征（BERT）。
- Decoder模型：仅包含解码器部分，适合自回归生成任务（GPT）。
- Encoder-Decoder模型：结合二者，适合序列到序列任务（T5）。

## BERT：Bidirectional Encoder Representation from Transformers

1. Encoder Only架构，仅使用Transformer编码层器。
2. 预训练过程：采用无监督学习，使用掩码语言模型和下一句预测进行训练。

   掩码语言模型（Masked Language Model）：在输入文本中随机掩盖部分词汇，模型需要预测被掩盖的词。

   下一句预测：训练模型判断两段文本是否为相邻句子，提高理解能力。
3. 微调过程。

## GPT：Generative Pretrained Transformer

1. Decoder Only架构，使用单向自回归模型，仅包含解码器层。
2. 预训练，采用自回归语言模型任务（Causal Language Modeling）基于前文预测下一个Token
3. 微调过程。主要用于文本升成任务，如对话生成，摘要生成。

## T5：Text-to-text Transfer Transformer

1. 完整Encoder-Decoder架构
2. 预训练：文本填空：类似Bert的MLM，但可以连续掩盖多个token。去噪：在输入文本中随机删除部分段落或单词，让模型恢复完整的文本。
3. 微调：适用于多种文本生成任务，包括文本摘要、翻译、文本分类。

# 3. 预训练

预训练（Pretraining）是指在大规模原始语料（一般是互联网级别数据）上，通过自监督任务（例如预测下一个词或 Masked LM）来学习语言规律。其目标是让模型获得通用的语言理解与生成能力，而非解决特定任务。

# 数据量级：从百万到万亿Token

预训练所需的数据量主要取决于模型规模（参数量）。

经验规律来自于 Scaling Laws（Kaplan et al., 2020）：模型越大，最佳训练 token 数也要按一定比例增加，否则会出现“欠训练”（undertrained）。

| 模型规模 | 代表模型 | 训练 Token 数（经验值） | 数据体量（估算） |
| --- | --- | --- | --- |
| ~100M 参数 | 小型语言模型（如DistilGPT） | $10^9$（1B） | 数十 GB |
| ~1B 参数 | GPT-2 Small | $10^{10}$（10B） | 百 GB |
| ~6B 参数 | GPT-J / GPT-NeoX | $10^{11}$（100B） | TB 级 |
| ~70B 参数 | LLaMA-2 / Falcon-70B | $2\times10^{12}$（2T） | 数十 TB |
| ~1T 参数 | GPT-4 级别 | $10^{13}$（10T） | 数百 TB |

- 经验比例规律：理想训练 Token 数 ≈ **20×参数量（tokens）**例如：一个 70B 模型理想上应训练约 1.4T tokens。

# 训练时间消耗

假设我们使用 A100 GPU（80GB）或 H100 GPU（80GB）：

| 模型 | GPU 类型 | GPU 数量 | 数据量 (Tokens) | 训练时间 (估算) |
| --- | --- | --- | --- | --- |
| GPT-2 (1.5B) | V100 × 32 | 32 GPUs | 40B | ~1 周 |
| GPT-J (6B) | A100 × 64 | 64 GPUs | 400B | ~2-3 周 |
| LLaMA-2 70B | A100 × 2048 | 2048 GPUs | 2T | ~3-4 周 |
| GPT-4 级 | H100 × 10,000+ | 数万 GPUs | 10T | 数月级 |

# 数据准备的工作量

预训练语料往往来源于混合多源数据：

| 数据类型 | 占比 | 来源 | 用途 |
| --- | --- | --- | --- |
| Web 文本 | 50-70% | Common Crawl, Wikipedia | 语言多样性 |

| Books / Papers | 10-20% | BooksCorpus, ArXiv | 长上下文、逻辑性 |
| Code | 10-20% | GitHub, StackOverflow | 程序理解 |
| Dialogue / QA | 5-10% | OpenAssistant, Reddit | 交互结构 |

数据清洗往往比训练本身还费时间。这些数据要经过：

- 去重（Dedup）；
- 语言检测；
- 垃圾过滤；
- 文本标准化（tokenize、normalize）。

# 4. 模型微调

## SFT（监督微调）与Continue Pretraining：

SFT 是一种常见的技术，用于在特定任务上微调预训练模型。这个过程通常需要构建一个高质量的指令微调数据集。

### 确定任务和指令

任务定义：明确你想要模型完成的具体任务，例如文本分类、情感分析或机器翻译。
指令构建：根据任务定义，构建相应的指令模板。例如对于文本分类任务，指令可能是“将以下文本分类为正面或负面”。

### 收集原始数据

数据来源：收集与任务相关的原始数据，这些数据可以来自公开数据集、专业数据库或众包平台。
数据清洗：对收集的数据进行清洗，去除无效或不相关的信息。

### 标注数据

标注标准：制定明确的标注标准，确保数据标注的一致性和准确性。

人工标注：通过人工标注或使用半自动化工具来标注数据，包括指令和预期输出。

## 设计指令微调数据集

数据分割：将数据集分割为训练集、验证集和测试集。

指令格式化：确保指令与数据格式一致。例如，使用JSON、CSV或特定格式的文本文件。

## 增强数据多样性

数据增强：通过同义词替换、句子重构等方法增加数据多样性，提高模型的泛化能力。

覆盖不同场景：确保数据集覆盖不同的使用场景和语境。

## 评估和迭代

初步评估：在初步构建的数据集上测试模型评估其性能。

迭代优化：根据评估结果，对数据集进行迭代优化，如增加更多样例或调整指令。

SFT让大模型在狭窄任务上过拟合人类偏好数据。从而使模型在通用性、语言多样性、探索能力上下降。解决的办法响应有混合instruction+Pretrain data，在SFT后添加DPO/ PPO/ ORPO 等偏好优化。

# 微调基础：

显存消耗 = 参数量 * 4 Bytes * 3（参数、梯度、优化器内部状态 ）

解决方案：

- 使用LoRA/QLoRA/Adapter等进行低秩微调
- 混合精度（FP16，BF16）
- 梯度累积
- ZeRO stage 2/3分布式优化。

# 数据构造与指令微调：

# 训练策略与优化技巧：

# 模型效果，泛化分析：

🔍 训练优化：从预训练到RLHF全流程

预训练阶段：万亿级语料（书籍/网页/代码）的清洗策略，无监督任务设计（MLM掩码语言模型、Causal LM自回归预测）；

微调技术：参数高效微调（LoRA的低秩适配、QLoRA的量化+低秩）对比全量微调的优劣；

RLHF核心：三阶段流程（SFT监督微调→RM奖励模型训练→PPO强化学习优化），奖励模型设计的关键指标（人类偏好对齐）。

-

⚙️ 工程实践：效率与性能优化

分布式训练：并行策略对比（数据并行解决样本量问题、模型并行拆分大参数量、流水并行优化长序列计算），ZeRO系列优化内存占用的核心逻辑；

量化压缩：INT4/INT8量化（GPTQ/AWQ算法）的精度损失控制，模型剪枝（结构化/非结构化）对推理速度的提升效果。

-

🚀 应用落地：Prompt与RAG技术栈

Prompt Engineering：Few-shot/Zero-shot提示设计原则，Chain-of-Thought（思维链）的逻辑推理引导方法，In-context Learning的上下文学习机制；

RAG检索增强：检索器（向量数据库如FAISS）与生成器（LLM）的协同流程，文档分块策略（语义窗口）对检索精度的影响。

-

🌟 前沿动态：MoE与多模态融合

MoE架构：混合专家模型的路由机制（Gate网络选择专家），稀疏激活对计算效率的提升（如GLaM的1.2T参数仅激活10%专家）；

多模态对齐：CLIP的图文对比学习（Image-Text Pair构建特征空间），FLAVA的单塔多模态理解范式（统一编码器处理多模态输入）。