Transformer 问题串讲

MLE 算法指北

2025 年 2 月 2 日

1 Transformer 架构及 Self-Attention 机制

Transformer 是由 Vaswani 等人于 2017 年提出的一种深度学习架构，广泛应用于自然语言处理（NLP）和计算机视觉（CV）领域。其核心是 **自注意力机制（Self-Attention）**，能够捕捉序列中远距离的依赖关系。

1.1 Transformer 架构概述

Transformer 由 **编码器-解码器（Encoder-Decoder）** 组成，具有以下主要模块：

- **编码器（Encoder）：**
  - 多头自注意力机制（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（Decoder）：**
  - Masked Multi-Head Self-Attention
  - 编码-解码注意力（Encoder-Decoder Attention）
  - 前馈神经网络

—

1.2 QKV 计算公式

在 Transformer 中，每个输入 token 通过一个线性变换映射到查询（Query）、键（Key）和值（Value），如下所示：

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

其中：

- $X \in \mathbb{R}^{n \times d}$ 是输入序列（n 表示序列长度，d 表示嵌入维度）。
- $W^Q, W^K, W^V \in \mathbb{R}^{d \times d_k}$ 是学习的权重矩阵，d_k 是注意力维度。
- 生成的 Q, K, V 分别用于计算注意力分数。

—

1.3 Self-Attention 机制

自注意力机制通过查询（Query）和键（Key）之间的点积计算注意力权重，并用这些权重加权值（Value），公式如下：

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

** 步骤解析：**

1. 计算注意力分数：

$$\text{score}_{ij} = \frac{q_i \cdot k_j^T}{\sqrt{d_k}}$$

其中 q_i 和 k_j 分别是第 i 个查询和第 j 个键，d_k 是缩放因子，防止梯度消失。

2. 通过 Softmax 归一化注意力权重：

$$\alpha_{ij} = \frac{\exp(\text{score}_{ij})}{\sum_{j=1}^{n} \exp(\text{score}_{ij})}$$

3. 计算最终输出：

$$z_i = \sum_{j=1}^{n} \alpha_{ij} v_j$$

** 特点：**

- 可以捕获序列中远距离依赖。
- 计算复杂度为 $O(n^2)$，适合并行化。

—

1.4 多头注意力机制（Multi-Head Attention）

在 Transformer 中，使用 **多头注意力**来从不同子空间关注信息，其公式为：

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

其中：

- h 为头数，每个注意力头有独立的权重 W_i^Q, W_i^K, W_i^V。
- 输出连接后再投影到最终维度 d。

作用：

- 允许模型关注输入的不同部分。
- 提高学习能力，增强模型对不同特征的理解。

—

1.5 Transformer 编码器流程

编码器部分的流程如下：

1. 输入嵌入：将输入序列转换为词向量，添加位置编码：

$$E_i = \text{Embedding}(x_i) + \text{PositionalEncoding}(i)$$

2. 多头自注意力：

$$Z = \text{MultiHead}(Q, K, V)$$

3. 前馈神经网络：

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

4. 残差连接 + 层归一化：

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

—

1.6 Transformer 解码器流程

解码器的流程与编码器类似，但增加了一个额外的 Encoder-Decoder Attention：

1. Masked Multi-Head Self-Attention
2. Encoder-Decoder Attention，计算如下：

$$\text{Attention}(Q, K_{\text{encoder}}, V_{\text{encoder}})$$

3. 前馈神经网络（FFN）

—

1.7 Self-Attention 机制的作用

Self-Attention 的主要作用包括：

- **长距离依赖捕获：**传统 RNN 处理长序列时，存在梯度消失问题，而 Self-Attention 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，大幅提高计算效率。
- **权重可视化：** Self-Attention 提供了解释性，可以可视化注意力权重，了解模型如何关注输入的不同部分。
- **处理变长输入：**不同于 CNN 需要固定大小的输入，Self-Attention 能灵活地处理可变长度的输入序列。

—

总结：

- Transformer 架构以 Self-Attention 为核心，取代了 RNN 中的时序依赖。
- QKV 计算通过线性变换获取不同表示，核心计算包括矩阵乘法和 softmax 归一化。
- Self-Attention 提升了远距离依赖捕捉能力，并支持并行计算。

2 BERT、GPT、T5 架构及其预训练与微调过程

近年来，基于 Transformer 的预训练语言模型（PLMs）在自然语言处理（NLP）领域取得了重大突破。BERT、GPT 和 T5 是最具代表性的三种模型，它们在架构和训练方式上有所不同，适用于不同的下游任务。

—

2.1 模型架构分类

根据 Transformer 模型的架构，预训练语言模型可以分为三类：

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

—

2.2 BERT: Bidirectional Encoder Representations from Transformers

1. 模型架构： BERT 采用 **Encoder-Only** 架构，即仅使用 Transformer 编码器层。

2. 预训练过程（Pretraining）：采用无监督学习，使用以下两种任务进行训练：

- **掩码语言模型（Masked Language Model, MLM）**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：

$$P(x_i|x_{1:i-1}, x_{i+1:n})$$

  示例：输入：`The cat is [MASK] the mat.` 目标：`on`。

- **下一句预测（Next Sentence Prediction, NSP）**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（Finetuning）： BERT 适用于文本分类、命名实体识别、文本匹配等任务，通常将下游任务头（如分类器）添加到最后一层隐藏状态。

—

2.3 GPT: Generative Pretrained Transformer

1. 模型架构： GPT 采用 **Decoder-Only** 架构，使用单向自回归（Auto-Regressive）模型，仅包含 Transformer 解码器层。

2. 预训练过程（Pretraining）：采用自回归语言模型任务（Causal Language Modeling, CLM），即基于前面的文本预测下一个 token：

$$P(x_i|x_{1:i-1})$$

示例：输入：`The cat is on the` 目标：`mat`。

3. 微调过程（Finetuning）： GPT 主要用于文本生成任务，如对话生成、摘要生成、代码生成等。

—

2.4 T5: Text-to-Text Transfer Transformer

1. 模型架构： T5 采用 **Encoder-Decoder** 架构，结合了编码器和解码器。

2. 预训练过程（Pretraining）： T5 统一 NLP 任务，将所有任务转换为文本到文本的形式。预训练采用以下任务：

- **文本填空（Text Infilling）**类似 BERT 的 MLM，但可以掩盖连续多个 token：

  Input: "The cat [MASK] the mat"

  Target: "is on"

- **去噪（Denoising Objective）**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. 微调过程（Finetuning）： T5 适用于多种文本生成任务，包括文本摘要、翻译、文本分类等。

2.5 预训练与微调过程对比

模型	预训练任务	微调任务	架构类型
BERT	MLM, NSP	分类、问答、NER	Encoder-Only
GPT	CLM（自回归）	文本生成、对话生成	Decoder-Only
T5	文本填空、去噪	摘要、翻译、分类	Encoder-Decoder

表 1: BERT、GPT、T5 预训练与微调过程对比

2.6 Encoder、Decoder、Encoder-Decoder 结构对比

- **Encoder（如 BERT）：**
  - 通过双向注意力学习上下文信息。
  - 适用于分类、信息提取等任务。
  - 局限：不适合生成任务。
- **Decoder（如 GPT）：**
  - 通过自回归方式生成序列。
  - 适用于对话、代码生成等任务。
  - 局限：难以处理双向上下文。
- **Encoder-Decoder（如 T5）：**
  - 结合编码器和解码器，支持序列到序列任务。
  - 适用于翻译、摘要等复杂任务。

2.7 优缺点比较

架构	优点	缺点
Encoder-Only	强大的表示能力，适合分类任务	不能直接生成文本
Decoder-Only	良好的文本生成能力	难以捕捉全局信息
Encoder-Decoder	适用于序列到序列任务	计算成本较高

表 2: Encoder, Decoder, Encoder-Decoder 对比

2.8 总结

- BERT 适合需要强上下文理解的任务，如文本分类和命名实体识别。
- GPT 适用于生成任务，如文本对话和内容创作。
- T5 作为通用架构，适合大多数 NLP 任务，尤其是序列到序列任务。

3 模型蒸馏与量化

随着深度学习模型的规模不断扩大（如 BERT、GPT 系列），模型的推理效率和部署难度成为瓶颈。模型蒸馏（Distillation）和模型量化（Quantization）是两种常用的模型压缩方法，能够在保持性能的同时显著减少模型大小和推理成本。

—

3.1 1. 模型蒸馏（Model Distillation）

定义：模型蒸馏是一种模型压缩技术，通过将一个大规模的预训练模型（称为 **教师模型**，Teacher）训练出的知识迁移到一个较小的模型（称为 **学生模型**，Student），使学生模型在性能上接近教师模型，同时显著减少模型参数量和计算成本。

—

3.1.1 1.1 蒸馏的核心原理

在模型蒸馏过程中，学生模型通过模仿教师模型的行为来学习知识。蒸馏损失由两部分组成：

- **硬标签损失（Hard Label Loss）：**学生模型需要对训练数据的真实标签进行优化。
- **软标签损失（Soft Label Loss）：**学生模型需要模仿教师模型的输出概率分布，捕捉教师模型的知识。

总损失函数：

$$\mathcal{L}_{\text{total}} = \alpha \mathcal{L}_{\text{hard}} + (1 - \alpha) \mathcal{L}_{\text{soft}}$$

其中：

$$\mathcal{L}_{\text{soft}} = -\sum_{i} P_{\text{teacher},i} \log P_{\text{student},i}$$

温度系数 T：为了提高蒸馏过程的鲁棒性，Softmax 函数引入温度 T 来平滑概率分布：

$$P_{\text{teacher},i} = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

—

3.1.2 1.2 蒸馏的优点

- 显著减少模型参数量和推理时间。
- 利用教师模型的知识，学生模型可以在小数据集上实现良好的泛化性能。
- 保持教师模型的准确性，同时显著降低资源消耗。

—

3.1.3 1.3 BERT 的蒸馏版本

由于原始 BERT 模型参数量大（BERT-base 110M 参数），多次推理需要较高计算成本，因此研究者提出了多种蒸馏版本的 BERT。

1. DistilBERT: - 通过蒸馏方法压缩 BERT 的参数量至 40%，推理速度提升约 60%。- 训练时保留 BERT 的结构，只减少层数（12 层减少到 6 层）。- 损失函数包含 MLM 的预测误差和蒸馏损失：

$$\mathcal{L} = \alpha \mathcal{L}_{\text{MLM}} + (1 - \alpha) \mathcal{L}_{\text{distill}}$$

—

2. TinyBERT: - 基于层对齐蒸馏，教师和学生在同层间匹配注意力分布和隐藏状态。- 包含两阶段训练过程：预训练蒸馏和微调蒸馏。- 能够适应多种任务，如文本分类、问答等。

—

3. MiniLM: - 强调蒸馏过程中对注意力矩阵的模仿：

$$\mathcal{L}_{\text{attention}} = \| QK/\sqrt{d_k} \|_{\text{teacher}} - \| QK/\sqrt{d_k} \|_{\text{student}}$$

- 训练简单，适合小型设备部署。

—

3.2 2. 模型量化（Model Quantization）

定义：量化是一种通过减少权重和激活值的精度来压缩模型的方法，例如将 32 位浮点数（FP32）转换为 16 位浮点数（FP16）或 8 位整数（INT8）。

—

3.2.1 2.1 量化类型

- **动态量化（Dynamic Quantization）：**在推理过程中对部分操作进行动态量化，例如仅对矩阵乘法进行量化。
- **静态量化（Static Quantization）：**在模型保存时就对权重和激活值进行量化，推理时直接使用低精度表示。
- **训练感知量化（Quantization-Aware Training, QAT）：**在训练过程中模拟量化误差，使模型在低精度下也能保持高精度推理性能。

—

3.2.2 2.2 量化公式

假设权重 W 的范围为 $[W_{\min}, W_{\max}]$，量化到整数范围 $[Q_{\min}, Q_{\max}]$ 的公式如下：

$$Q = \text{round}\left(\frac{W - W_{\min}}{W_{\max} - W_{\min}} \cdot (Q_{\max} - Q_{\min})\right)$$

反量化公式：

$$W_{\text{dequantized}} = Q \cdot \frac{W_{\max} - W_{\min}}{Q_{\max} - Q_{\min}} + W_{\min}$$

—

3.2.3 2.3 量化的优点

- 显著减少模型大小（FP32 转 INT8 可减少 75%）。
- 提升推理效率，尤其在边缘设备上性能显著。
- 支持无额外训练直接量化（动态量化）。

—

3.3 3. 模型蒸馏与量化对比

方法	优点	缺点
模型蒸馏	高效压缩模型，保留性能	需额外训练，依赖教师模型
模型量化	显著减少模型大小与推理时间	低精度可能损失性能

表 3: 模型蒸馏与量化的对比

—

3.4 4. 总结

- 模型蒸馏适合需要高精度的小型模型场景，例如移动设备上的 NLP 应用。
- 模型量化适合资源受限的场景，例如边缘计算设备。
- BERT 的蒸馏版本（如 DistilBERT、TinyBERT）展示了蒸馏在大模型压缩中的强大效果。