

## 手工实现并分析小规模 Transformer

学生 张子烨

学号 25125234

## 摘要

本报告实现了一个基于 Transformer 架构的仅解码器（Decoder-Only）语言模型，并在 Tiny Shakespeare 数据集上进行了训练。为了探究 Transformer 架构中各个核心组件的必要性，本报告执行了一系列严格的消融实验（Ablation Study）。实验系统地移除了三个关键部分：**位置编码（Positional Encoding）**、**残差连接（Residual Connections）** 和 **多头注意力机制（Multi-Head Attention）**（通过将其简化为单头注意力）。本报告详细阐述了基准模型（Baseline Model）的数学原理、实验设置以及伪代码实现，并对消融实验的预期结果进行了分析。实验结果（如图表所示）将量化这些组件对模型学习能力和最终性能的贡献。

## 1 引言

近年来，基于 Transformer 架构的大型语言模型（LLM）在自然语言处理（NLP）领域取得了革命性的进展。从开创性的“Attention Is All You Need”论文到 GPT 系列等仅解码器模型的广泛应用，Transformer 已成为序列建模的标准。

Transformer 的成功归功于其复杂的架构设计，其中包括自注意力机制、多头分解、位置编码、残差连接和逐层归一化等。然而，这些组件各自的贡献和必要性是一个值得深入研究的问题。一个组件是提供了微小的性能提升，还是对模型的收敛能力至关重要？

为了解答这个问题，本报告采用**消融研究**的方法。首先构建一个功能完整的仅解码器（Causal Decoder）语言模型作为基准（Baseline）。随后，创建该模型的三个“残缺”版本：

1. **移除位置编码**: 测试模型在缺乏显式序列顺序信息下的表现。
2. **移除残差连接**: 探究其在缓解梯度消失和促进深度网络训练中的作用。
3. **使用单头注意力**: 评估多头机制（即将信息分解到不同表示子空间）的优势。

本报告将详细介绍模型的技术实现，包括其核心的数学推导，并通过伪代码展示训练流程。

## 2 相关工作

本研究建立在几个关键的先前工作之上：

- **Transformer (Vaswani et al., 2017)**: 首次提出了完全基于注意力的序列到序列模型，摒弃了循环和卷积结构。本报告的模型大量借鉴了其编码器-解码器架构中的解码器部分。
- **生成式预训练 Transformer (GPT) (Radford et al., 2018, 2019)**: 验证了仅解码器 Transformer 在大规模无监督数据上预训练后，在下游任务上进行微调的

巨大潜力。基准模型在架构上与 GPT 的早期版本相似，均采用因果掩码（Causal Masking）进行自回归生成。

- **模型理解与消融研究:** 对复杂神经网络（如 ResNet, BERT 等）进行消融研究是理解其内部机制的常用方法。这些研究通过系统性地移除或替换模型组件，来衡量该组件对整体性能的影响。

## 3 模型架构与数学原理

基准模型是一个堆叠了  $N$  层的仅解码器（Decoder-Only）Transformer。其核心组件的数学原理如下。

### 3.1 位置编码 (Positional Encoding)

由于自注意力机制本身不包含序列顺序信息，使用固定的正弦/余弦位置编码  $PE \in \mathbb{R}^{\text{max\_len} \times d_{\text{model}}}$ 。对于任意位置  $pos$  和维度索引  $i$ ，其定义如下：

$$\begin{aligned} PE(pos, 2i) &= \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \\ PE(pos, 2i + 1) &= \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \end{aligned} \quad (1)$$

此  $PE$  矩阵与词嵌入  $X_{\text{emb}}$  逐元素相加 ( $X_{\text{emb}}$  经过  $\sqrt{d_{\text{model}}}$  缩放后)，将位置信息注入模型。

### 3.2 多头因果自注意力 (Multi-Head Causal Self-Attention)

给定输入序列  $X \in \mathbb{R}^{B \times T \times d_{\text{model}}}$  ( $B$  为批量大小,  $T$  为序列长度)，多头注意力首先将其线性投影到查询 ( $Q$ )、键 ( $K$ ) 和值 ( $V$ )：

$$Q = XW^Q = XW^K = XW^V \quad (2)$$

其中  $W^Q, W^K, W^V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$  是可学习参数。这三个矩阵被重塑并转置，以分离  $h$  个注意力头，每个头的维度  $d_k = d_{\text{model}}/h$ 。核心的缩放点积注意力计算（在您的代码中由 `scaled_dot_product_attention` 实现）：

$$\text{Attention}(Q, K, V, M_{\text{causal}}) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M_{\text{causal}}\right)V \quad (3)$$

关键在于因果掩码  $M_{\text{causal}}$ （由 `generate_causal_mask` 生成）：

$j > i$	$-\infty$
---------	-----------

（在代码中为 `-1e9`），确保在  $t$  时刻的预测只能关注到  $t$  时刻及之前的位置。各头输出被拼接（Concat）并经过最终的线性投影  $W^O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ ：

$$\text{MultiHead}(X, M_{\text{causal}}) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \quad (4)$$

### 3.3 位置前馈网络 (Position-wise FFN)

位置前馈网络 (FFN) 独立地应用于序列中的每个位置。它由两个线性层和一个 ReLU 激活函数组成 (根据您的 PositionWiseFeedForward 类):

$$\text{FFN}(x) = \text{ReLU}(xW_1)W_2 \quad (5)$$

其中  $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$ ,  $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$ 。中间层维度  $d_{\text{ff}}$  通常大于  $d_{\text{model}}$ 。

### 3.4 残差连接与 LayerNorm (Post-LN)

您的 CausalDecoderBlock 实现遵循了原始 Transformer 论文中的 Post-LN (后归一化) 结构。每个块包含两个子层 (多头注意力和 FFN)，均采用残差连接、Dropout 和层归一化。对于一个块的输入  $x$ ，其计算流程为:

$$\begin{aligned} x' &= x + \text{Dropout}(\text{MultiHead}(x, x, x, M_{\text{causal}})) \\ \hat{x} &= \text{LayerNorm}(x'') \\ &= \hat{x} + \text{Dropout}(\text{FFN}(\hat{x})) = \text{LayerNorm}(x'') \end{aligned} \quad (6)$$

其中  $y$  是该块的最终输出，并作为下一块的输入。残差连接 ( $x + \dots$  和  $\hat{x} + \dots$ ) 是构建深度网络的关键，它允许梯度在反向传播中“直通”层，有效缓解了梯度消失问题。

## 4 实验设计与伪代码

使用 PyTorch 框架实现了模型。为了优化训练过程，采用了 AdamW 优化器 (一种结合了  $L_2$  正则化/权重衰减 (Weight Decay) 的 Adam 变体)，并配合余弦退火学习率调度器 (Cosine Annealing LR Scheduler)。该调度器使学习率在训练过程中平滑地从初始值下降到最小值 ( $\eta_{\min}$ )，有助于模型在训练后期收敛到更优的局部最小值。

### 4.1 训练循环伪代码

```
训练的核心逻辑在 train_one_epoch

model, optimizer, scheduler, train_data, bptt, device, vocab_size
average_epoch_loss

total_loss = 0
num_batches = 0
model.train()
for i = 0 to (train_data.size(0) - 1) step 1
    data, targets = get_batch(train_data, i, bptt)
    seq_len = data.size(1)

    // 2. 生成因果掩码 (S, S)
```

```

mask = generate_causal_mask(seq_len, device)

// 3. 前向传播
optimizer.zero_grad()
output = model(data, mask) // (B, S, V)

// 4. 计算损失 (忽略 PAD_TOKEN)
loss = CrossEntropyLoss(output.view(-1, V),
                        targets.view(-1),
                        ignore_index=PAD_TOKEN)

// 5. 反向传播与优化
loss.backward()
torch.nn.utils.clip_grad_norm_(model.parameters, max_norm=0.5)
optimizer.step()

total_loss += loss.item()
num_batches += 1
// 6. 在周期结束后更新学习率scheduler.step()return total_loss / num_batches

```

## 4.2 消融实验实现

通过对基准模型 (Baseline Model) 进行深度复制 (deepcopy) 并修改其关键组件，来创建三个消融实验版本：

1. **无位置编码 (Ablation<sub>NoPositionalEncoding</sub>) : Identity**

```
model_no_pe.pos_encoding = nn.Identity().to(device)
```

这使得加法操作  $x = x + \text{self.pe}[\dots]$  变为  $x = x + 0$ ，从而完全移除了位置信息。

### 无残差连接 (Ablation\_No\_Residuals):

定义一个新的 NoResidualCausalDecoderBlock 类，它继承自 CausalDecoderBlock 但重写了其 forward 方法。原始的残差连接被移除：

原始 (Post-LN):  $\hat{x} = \text{LayerNorm}(x + \text{Dropout}(\dots))$

消融:  $\hat{x} = \text{LayerNorm}(\text{Dropout}(\dots))$

模型中的所有 CausalDecoderBlock 实例均被替换为这个无残差的版本。

### 单头注意力 (Ablation\_Single\_Head):

此消融通过在实例化 DecoderOnlyLanguageModel 时，将超参数 n\_heads 从 4 (基准值) 修改为 1 来实现。

```
model_single_head = DecoderOnlyLanguageModel(..., n_heads=1, ...)
```

这迫使模型在单个表示子空间中执行所有注意力计算，而不是将  $d_{\text{model}}$  分解为  $h$  个独立的头。

## 5 预期结果与分析

基于所提供的代码实现和 Transformer 的既有理论，预期在图表中将观察到以下不同模型的验证损失（Validation Loss）曲线：

### 1. 基准模型 (Baseline)

- **预期结果:** 将表现最佳。其验证损失曲线将稳步下降，并收敛到四个实验中的最低水平。
- **分析:** 这证明了模型的所有组件——位置编码、多头注意力、残差连接和 FFN——协同工作时的有效性，为其他实验提供了性能基准。

### 2. 无残差连接 (Ablation: No Residuals)

- **预期结果:** 将表现最差。模型的训练将极其困难，验证损失可能完全无法收敛（即损失保持在一个非常高的水平，几乎不下降），甚至可能发散（由于梯度爆炸/消失导致损失变为 NaN）。
- **分析:** 即使对于一个仅 2 层 ( $N\_LAYERS=2$ ) 的浅层网络，移除残差连接也会严重阻碍梯度在层间的有效反向传播。这凸显了残差连接是训练（哪怕是浅层）Transformer 网络的必要条件。

### 3. 无位置编码 (Ablation: No Positional Encoding)

- **预期结果:** 表现将非常差，其验证损失将远高于基准模型，但可能仍优于“无残差连接”的版本。
- **分析:** 缺乏位置信息，模型将退化为一种“词袋”(Bag-of-Words) 模型。它无法区分“A B”和“B A”的语义差异，只能依赖词元共现来预测。这证明了位置信息对于理解语法和序列顺序是极端重要的。

### 4. 单头注意力 (Ablation: Single Head)

- **预期结果:** 表现将差于基准模型，但显著优于“无位置编码”和“无残差连接”的版本。
- **分析:** 模型仍然可以学习序列信息（因为它有位置编码）并且可以训练（因为它有残差连接）。然而，它失去了“多头”机制的优势——即在  $h$  个不同的表示子空间 (representation subspaces) 中并行捕捉信息的能力。单头被迫在单一空间中混合所有信息，降低了模型的表达能力。这证明了多头分解是一个显著有效的架构改进。

**总结：**预期的实验结果将清晰地展示，残差连接对于模型能否训练至关重要，位置编码对于模型能否理解顺序至关重要，而多头注意力则为模型提供了更强的表达能力。

## 6 复现指南

### 6.1 环境与依赖

本项目基于 Python 3.x 实现，核心依赖库为 PyCharm。推荐使用虚拟环境进行管理。

#### 1. 克隆仓库

代码存放在github中，链接如下：<https://github.com/LlberC/transformer-ablation>

```
git clone https://github.com/LlberC/transformer-ablation.git
```

#### 2. 创建并激活虚拟环境

(我以 pycharmEnv 为例，您可以按需命名)

##### 1. 创建环境

```
python -m venv pycharmEnv
```

##### 2. 激活环境（根据您的终端）

Windows (CMD)

```
.\pycharmEnv\Scripts\activate.bat
```

Windows (PowerShell) - 您可能需要先运行 Set-ExecutionPolicy

```
.\pycharmEnv\Scripts\Activate.ps1
```

Windows (Git Bash) - 这是您在 PyCharm 中最常用的

```
source ./pycharmEnv/Scripts/activate
```

#### 3. 安装依赖

(请确保您已处于已激活的虚拟环境中，提示符前有 (pycharmEnv))

```
pip install -r requirements.txt
```

### 6.2 数据准备

下载 Tiny Shakespeare 数据集。

#### 1. (Windows 用户) 如果您使用 Git Bash，请先赋予脚本权限：

```
chmod +x scripts/prepare_data.sh
```

## 2. 运行脚本:

```
bash scripts/prepare_data.sh
```

该脚本会检查 `input.txt` 是否存在。如果您已将该文件手动放置在项目根目录，脚本将自动跳过下载。

## 6.3 运行实验

建议使用 PyCharm 的“运行配置”功能来管理和运行实验。

### 1. 创建实验配置:

- 按照之前的讨论，创建 4 个 Python 运行配置，全部指向 `src/main.py` 脚本。
- **重要：**确保所有配置的 Working directory (工作目录) 都指向您的项目根目录 (例如 `C:...-ablation`)。
- **重要：**确保所有配置的 Python interpreter (解释器) 都指向您 pycharmEnv 中的 `python.exe`。
- 在 Parameters (参数) 字段中分别填入：
  - 配置 1 (Name: `baseline`): `-experiment baseline -seed 42`
  - 配置 2 (Name: `no_pe`): `-experiment no_pe -seed 42`
  - 配置 3 (Name: `no_res`): `-experiment no_res -seed 42`
  - 配置 4 (Name: `single_head`): `-experiment single_head -seed 42`

### 2. 运行实验:

在 PyCharm 顶部的下拉菜单中，依次选择并运行这 4 个配置。

## 6.4 结果查看与绘图

所有实验产出物都将保存在 `results/` 目录中。

- `results/{exp_name}_metrics.json`: 每个实验的损失数据。
- `results/{exp_name}_best_model.pth`: 每个实验保存的最佳权重。

要生成或更新最终的汇总图表 (`ablation_study_loss_curve.png`)，您必须在所有 4 个实验之后运行绘图脚本：

### 1. PyCharm (推荐):

运行您创建的 Plot Results 配置 (它指向 `src/plot_all.py`，无参数，工作目录为根目录)。

### 2. 终端:

(确保环境已激活)

```
python src/plot_all.py
```

最终的对比图表将保存在: `results/ablation_study_loss_curve.png`。

## 7 结论与展望

本次作业完成了从零实现 Transformer Encoder、构建训练流水线、在小数据集上验证性能，并通过消融实验分析位置编码与模型容量的影响。