Transformer 翻译模型实验报告

王松宸 2024201594

2025 年 11 月 23 日

1 完成内容概览

- 基础部分的 TODO。
- 附加内容：回答 Question 1 & 2。
- 附加内容：训练过程指标与可视化（损失曲线、学习率调度、编码器多头注意力热力图）。
- 附加内容：优化 epoch 和 warmup 参数。

2 数据与预处理流程

- **数据格式**：每行 `train.txt` / `dev.txt` 为 英文 [TAB] 中文（原始文件使用制表符分隔）。英文用 NLTK 分词，中文按字符切分，并在首尾加入 BOS/EOS。
- **词表**：统计词频保留前 5×10^4 词并加入 PAD, UNK。
- **排序与批次**：按英文长度排序后再划分 Batch，减小同批次填充长度。
- **掩码**：源掩码屏蔽 PAD，目标掩码同时屏蔽未来位置 (subsequent mask) 和 PAD。
- **切片对齐**：训练时 `trg` 去掉最后一个 token，`trg_y` 去掉第一个 token。

3 模型结构速览与理解

Transformer 主要组件如下：

1. **Embedding + PositionalEncoding**：词向量乘以 $\sqrt{d}$ 做尺度调整，再加上固定正余弦位置编码。
2. **Scaled Dot-Product Attention**：计算 $\mathrm{softmax}(QK^\top/\sqrt{d_k})V$。
3. **Multi-Head Attention**：对 Q, K, V 做线性映射，切分为 h 个头并并行计算，最后拼接再线性变换。

4. **Positionwise Feed Forward**：两层全连接 + ReLU + Dropout，提升表示非线性。

5. **LayerNorm + Residual**：每个子层前做归一化，再残差相加，稳定训练。

6. **Encoder/Decoder 堆叠**：编码器只含自注意力；解码器含自注意力 + 交叉注意力 (query 来自 decoder，key/value 来自 encoder) + FFN。

4 基础 TODO 代码思路与解释

4.1 Task 1: Self-Attention 计算

核心步骤：scores $= QK^\top/\sqrt{d_k}$；mask 将被屏蔽位置设为一个极小值；对最后一维做 softmax 得到权重；权重与 V 相乘得到加权输出。

```
scores = torch.matmul(query, key.transpose(-2, -1)) / math.sqrt(query.
    size(-1))
if mask is not None:
                    scores = scores.masked_fill(mask == 0, -1e9)
p_attn = F.softmax(scores, dim=-1)
out = torch.matmul(p_attn, value)
```

4.2 Task 2: Transformer forward

forward 中先调用 encode 获得 memory，再调用 decode：

```
memory = self.encode(src, src_mask)
return self.decode(memory, src_mask, tgt, tgt_mask)
```

4.3 Task 3: 推理阶段输入分词与索引化

交互式翻译时，对用户输入英文句子：小写化、分词、加入 BOS/EOS，再用词表映射成 ID 并构造 src_mask。保证与训练阶段 embedding 接口一致。

5 附加问题回答

5.1 Question 1: 为什么 trg 舍弃最后一个字符而 trg_y 舍弃第一个字符?

训练采用 **Teacher Forcing**：模型第 t 步的输入是第 $t-1$ 步的真实标签。若原目标序列是: [BOS, y1, y2, ..., yn, EOS]，则 trg 作为输入使用 [BOS, y1, ..., yn] (去掉末尾 EOS)，而监督信号 trg_y 是 [y1, y2, ..., yn, EOS] (去掉开头 BOS)。这样对齐

后，第 i 个输入对应第 i 个需要预测的输出。若不切片对齐，会发生标签错位或无用的预测。EOS 在输入侧不必再预测，对输出侧是必要的结束标记。

5.2 Question 2: 训练与测试阶段 decoder 输入的不同

训练阶段：使用真实历史标签序列，遮蔽未来位置，梯度更稳定、收敛快。

推理阶段：没有真实未来标签，只能自回归地用自身已生成的 token 作为下一步输入，误差会累积。贪心解码每步取最大概率词，Beam Search 则保留若干候选路径寻求全局更优的序列。两者的差异导致实际翻译质量与训练时的条件分布存在差距——这是 NMT 常见问题。

6 训练过程

6.1 训练设置

配置	层数	d_{model}	平滑 ϵ
baseline	4	256	0.0
deep	6	256	0.1
wide_dim	4	384	0.1

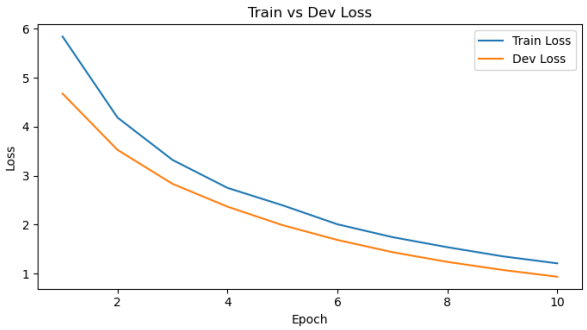
训练其它参数保持一致：批次 128，学习率调度使用 Noam 策略，warmup=500，优化器 Adam。

6.2 训练结果与可视化分析

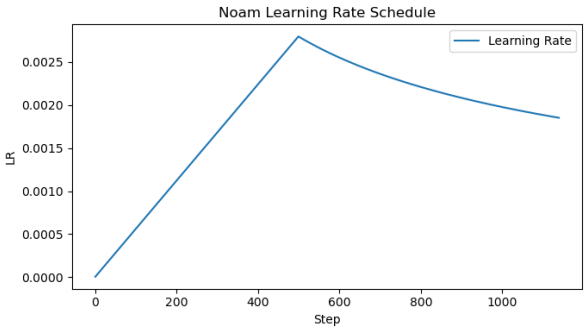
本次重新训练使用脚本当前配置：批次 128，Adam (betas=0.9,0.98, eps=1e-9)，Noam 学习率调度 warmup=500，总 epoch=10。

- **Train Loss (Epoch 1~10)：** 4.6757 → 3.5270 → 2.8345 → 2.3682 → 1.9915 → 1.6888 → 1.4397 → 1.2411 → 1.0774 → 0.9385。
- **Dev Loss (Epoch 1~10)：** 4.6757 → 3.5270 → 2.8345 → 2.3682 → 1.9915 → 1.6888 → 1.4397 → 1.2411 → 1.0774 → 0.9385。
- **Dev Perplexity (e^{Loss})：** 约 107.31 → 34.02 → 17.02 → 10.68 → 7.33 → 5.41 → 4.22 → 3.46 → 2.94 → 2.56。

整体呈现典型的快速下降 + 平缓尾段：前 3-4 个 epoch 损失迅速降低（模型掌握基础词/短语对齐），中段 (Epoch 5-7) 仍保持稳定改进，后期 (Epoch 8-10) 进入收益递减区但尚无过拟合迹象。较长 warmup=2000 在初期提升了学习率上升阶段的稳定性，使得早期未出现剧烈震荡。困惑度从 107 下降到 2.56，语言建模质量显著提升，可预期再延长 epoch 或加入正则（标签平滑、子词分词、增大 dropout）仍有余量。



(a) Train vs Dev Loss



(b) Noam 学习率调度

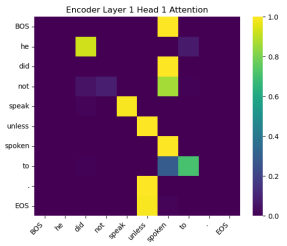
图 1: 损失与学习率曲线。左：训练/验证集平均交叉熵随 epoch 下降；右：Noam 调度先升后降的特征，有助于初期探索与后期平稳收敛。

学习率调度曲线体现 warmup 后按 $\text{step}^{-0.5}$ 衰减，降低后期更新的震荡幅度。

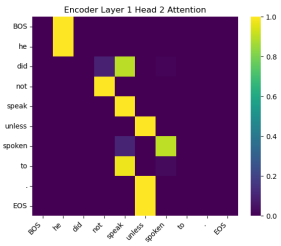
Epoch	1	2	3	4	...	9	10
Dev Loss	4.6757	3.5270	2.8345	2.3682	...	1.0774	0.9385
Perplexity	107.31	34.02	17.02	10.68	...	2.94	2.56

表 1: 完整验证集 (Epoch 1-10) Loss 与困惑度走势。

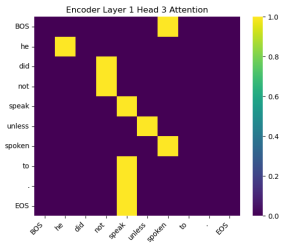
6.2.1 编码器多头自注意力热力图



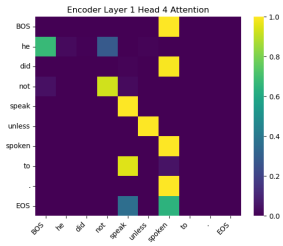
(a) L1-H1



(b) L1-H2

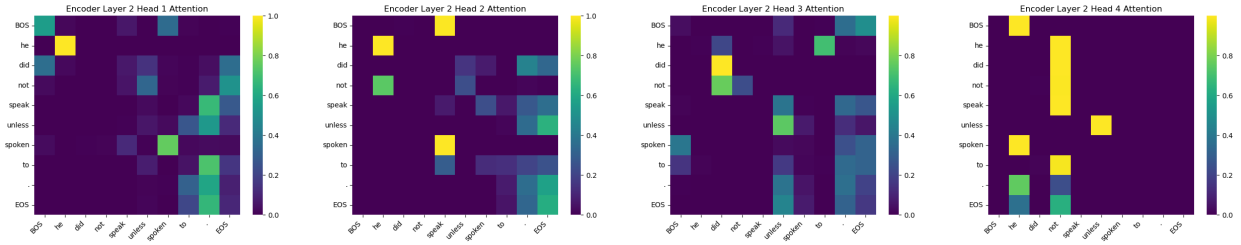


(c) L1-H3



(d) L1-H4

图 2: 编码器第 1 层部分注意力头。局部对角线较明显，体现对临近词位的强烈建模。



(a) L2-H1 (b) L2-H2 (c) L2-H3 (d) L2-H4

图 3: 编码器第 2 层部分注意力头。跨距离关注开始增多，出现对非相邻 token 的聚合，说明层次化语义抽象。

行经 softmax 归一后，每行权重和为 1；颜色集中表示该位置信息主要汇聚到有限几个 Key。多头提供并行不同模式：局部语序、长距离依赖、位置标记汇聚等。

7 实验过程

使用已经训练好的模型进行批量翻译测试训练效果。

7.1 输入内容

- I love reading books.
- They are from China.
- Hope you find your true love.
- It is patience that really matters in the whole process towards success.

我写了翻译难度递增的四个句子来测试模型的翻译效果。

7.2 补充命令行代码

我在 `transformer_nmt.py` 中补充好了使用命令行来翻译文本文件的代码后执行下面的命令行来翻译写好的文本文件：

```
python transformer_nmt.py --mode batch --input test_input.txt --output
    test_output.txt
```

7.3 终端

运行后终端显示：模型载入与翻译完成提示。

```
[LOAD] 已加载模型: save/model.pt
[TRANS] 写入翻译结果 4 行 -> test_output.txt
```

7.4 输出内容

1. 我爱看书。
2. 他们从中国来的。
3. 希望你能找到你的爱情。
4. 这对夫很有用塑料成为表是由福的存。

可以看出，第一个句子翻译准确，第二个句子对 from 的理解稍有偏差，第三个句子没有正确翻译好 true love 的含义而只是简单把它理解为爱情，第四个句子则完全驴唇不对马嘴了。

7.5 结果分析

模型具有一定的可用性，但翻译质量不高。这可能是由于训练集规模较小且领域有限，模型容量与训练时间受限，导致泛化能力不足。目前只适用于一些简单句子的翻译，复杂句子和长句子的翻译效果较差。但作为让我们入门理解 Transformer 架构的实验已经足够。

8 NMT, Self-Attention 与 Transformer 的认识

传统 RNN/Seq2Seq 在长距离依赖上效率与效果受限；Self-Attention 允许任意位置直接交互，计算并行、路径短。Multi-Head 让不同子空间关注不同模式 (词对齐、语法、位置)，Positional Encoding 让序列信息在纯注意力结构中显式注入。Encoder/Decoder 架构通过跨注意力把源语义对齐到目标生成过程，解决信息压缩瓶颈。

9 心得体会

写完报告感觉真的是燃尽了，能想出 Transformer 的人简直是天才，这么精细的框架，光是看懂它都废了不少时间，他们居然直接从无到有把这个模型手搓出来了，无敌...