作业一

Jerry(2500011484)

2026 年 1 月 31 日

1 Unicode

1.1 Problem : Unicode1

- 1. `'\x00'`
- 2. print 的结果为"",string representation 使用转义符号表示
- 3. python print 时直接忽略这个字符, 在 string representation 中仍然以`'\x00'`存在

1.2 Problem : Unicode2

- 1. UTF-8 将 unicode 转为`uin8_t`序列, 词汇表大小为 256, 比 UTF-16 和 UTF-32 更小
- 2. " 你好"

  解释: 因为这个函数是对 utf-8 编码逐字解码, 对于 utf-8 中的多字符编码 (ascii 以外), 会进行无效的解码
- 3. 0xffff 对于 utf-8, 这是长度为二的项, 要求其字符一的第六七位为 1, 第五位为 0, 然而 0xffff 的第五位为 1

2 BPE

2.1 Problem : BPE Training on TinyStories

- 1.(单线程)

  执行时间: 392.0504 秒

  最大内存使用: 110.32 MB
- 2.pre-tokenization 过程
- 3. 最长 token:" accomplishment", 是最长的单词

2.2 Problem : BPE Training on OpenWebText

- 1.(单线程)

  执行时间: 26135.2360 秒

  最大内存使用: 10959.24 MB (Linux)

- 2. 最长 token:'ÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ'(文中最长的" 词")

- 3. 有更多的词汇

2.3 Problem : Experiments with tokenizers

- 1. 压缩率

  TinyStories:4.36

  OpenWebText:4.32

- 2. 压缩率会下降, 甚至小于 1(由于不匹配导致必须使用前 256 个 vocab)

- 3. 速度在 8×10^4 到 1×10^5(bytes/second), 保守估计需要 412.5 小时 (不会多线程)

- 4. 其一大于 32000, 其二是 2 个字节对齐

3 Full Transformer LM

3.1 Problem : Transformer LM resource accounting

- 1. 大约共 15.57 亿个参数, 内存占用约 5.8GB

- 2. 大约 3.5×10^{12}FLOPS

  (1). 每个 Transformer 层（共 48 层）

  自注意力模块:

  Q,K,V 投影：将输入 X 分别乘以权重矩阵 W_Q, W_K, W_V，得到 Q,K,V。

  FLOPs:$6 \times n \times d^2$

  注意力分数计算：每个头的 Q_i 与 K_i^T 相乘，得到注意力分数矩阵。

  FLOPs:$2 \times n^2 \times d$（所有头合计，$d = h \times d_k$）。

  注意力加权和：每个头的注意力权重矩阵与 V_i 相乘。

  FLOPs:$2 \times n^2 \times d$（所有头合计）。

  输出投影：多头输出拼接后乘以权重矩阵 W_O。

  FLOPs:$2 \times n \times d^2$。

  前馈网络（FFN）模块：

  第一个线性层：输入乘以权重 W_1。

FLOPs:$2 \times n \times d \times d_f f$。

第二个线性层：中间激活乘以权重 W_2。

FLOPs:$2 \times n \times d_f f \times d$。

每个 Transformer 层的矩阵乘法总 FLOPs 为：

$(8 \times n \times d^2) + (4 \times n^2 \times d) + (4 \times n \times d \times d_f f)$

(2). 输出层（语言模型头）

词表投影：最后一层隐藏状态乘以输出权重矩阵 $W_o ut$。

FLOPs:$2 \times n \times d \times vocab_s ize$。

- 3. 每个 Transformer 层中的前馈网络（FFN）
- 4. 随着模型尺寸增加，前馈网络（FFN）的 FLOPs 占比显著上升（从小型到大型：39.76% → 49.85% → 54.46%），而自注意力核心计算（QK^T 和 AV）和输出层的占比下降（自注意力核心：13.25% → 12.46% → 10.89%；输出层：27.10% → 12.75% → 7.42%）。这是因为 FFN 的计算量随 d_{model} 的平方增长，而自注意力核心计算仅随 d_{model} 线性增长，导致前者占主导。
- 5. 当 GPT-2 XL 的上下文长度从 1024 增加到 16384 时，总 FLOPs 从约 3.507×10^{12} 增加到约 1.334×10^{14}。模型各部分的 FLOPs 贡献比例发生显著变化：自注意力核心计算（QK^T 和 AV）的占比从 9.19% 急剧上升到 61.81%，成为主要计算瓶颈；前馈网络（FFN）的占比从 57.42% 下降到 24.14%；线性投影部分占比从 28.71% 下降到 12.07%；输出层占比从 4.70% 下降到 1.98%

4 Optimizer

4.1 Problem : Tuning the learning rate

1e1 训练较为缓慢 (10iteration 导致 loss 降低到 4)
1e2 训练快:10iteration loss 降低到 10^{-23} 量级
1e3 训练导致发散 (步长过大)

4.2 Problem : Resource accounting for training with AdamW

- 1.

$$B = \text{batch\_size} \tag{1}$$
$$V = \text{vocab\_size} \tag{2}$$
$$L = \text{context\_length} \tag{3}$$
$$N = \text{num\_layers} \tag{4}$$
$$D = \text{d\_model} \tag{5}$$
$$H = \text{num\_heads} \tag{6}$$
$$d_{ff} = 4D \tag{7}$$

1. 参数数量:
   (a) Input embedding: VD
   (b) m-head: VD
   (c) 每层 Transformer（权重矩阵部分）:
      i. Q,K,V 投影：$3D^2$
      ii. attention output projection：D^2
      iii. FFN W1：$4D^2$
      iv. FFN W2：$4D^2$
   (d) 每层 RMSNorm 参数：$2D$
   (e) final RMSNorm（层外）：D
   (f) 总体参数量（float 数）:

$$P = 2VD + 12ND^2 + (2N+1)D$$

   (g) 字节数:

$$\text{Params\_bytes} = 4 \cdot P = 4(2VD + 12ND^2 + (2N+1)D)$$

2. 激活
   (a) RMSNorm(s)：$2 \cdot (BLD)$
   (b) Multi-head attention:
      i. Q,K,V 投影输出：$3 \cdot (BLD)$
      ii. QK 矩阵乘积：$B \cdot H \cdot L \cdot L$
      iii. softmax 后权重：$B \cdot H \cdot L \cdot L$
      iv. attention 加权和：BLD
      v. output projection 后：BLD
   (c) Position-wise FFN:
      i. W1 输出: $BL(4D)$

ii. SiLU 激活: $BL(4D)$

iii. W2 输出: BLD

iv. final RMSNorm：(B L D)

(d) output embedding：BLV

(e) cross-entropy on logits 保守计为 BLV

(f) 激活总浮点数：

$$A = BL((16N+1)D + 2V) + 2NBHL^2$$

(g) 对应字节数：

$$\text{Activations\_bytes} = 4 \cdot A = 4(BL((16N+1)D + 2V) + 2NBHL^2)$$

3. 梯度

$$\text{Gradients\_bytes} = 4 \cdot P$$

4. 优化器状态（AdamW）

$$\text{OpttState\_bytes} = 2 \cdot (4 \cdot P) = 8 \cdot P$$

5. 峰值总内存

$$16(2VD + 12ND^2 + (2N+1)D) + 4(BL((16N+1)D) + 2V) + 2NBHL^2$$

- 2.

  $\text{Memory}(K) = (5.44k + 105.2)GB$

  最大 batch size 为 0

- 3. `FLOPs/step`约为 $6BLN(12D^2)$

- 4. 训练时间约为 440days

5 Experiment

```
self.lm_head.weight = self.token_embeddings.weight
```

我在实际训练中将`lm_head`与`token_embedding`共享了权重

5.1 Problem : Experiment logging

```
print(
    f"epoch {epoch + 1} | "
    f"val loss {check_loss.item():.4f} | "
    f"std {check_logits.std().item():.4f} | "
    f"lr {current_lr:.2e}"
)
```

```
batch_time_loss_list.append([
    epoch,
    datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
    check_loss.item()
])

```

5.2 Problem : Tune the learning rate

- 1

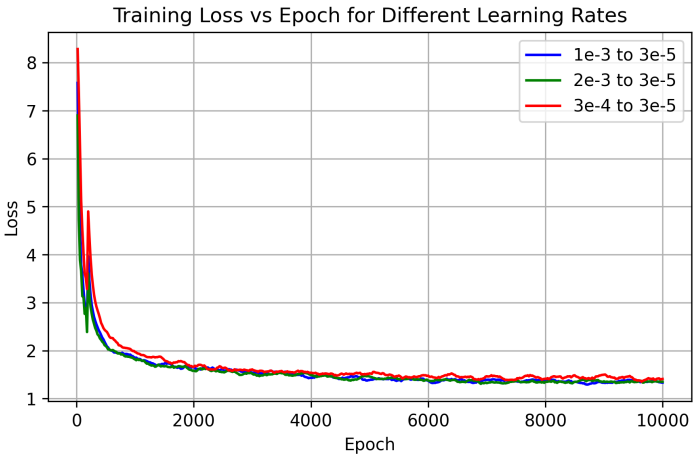
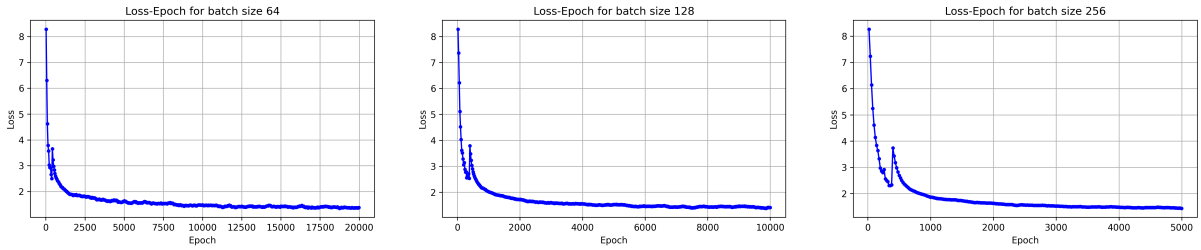


图 1: tunning learning rate

显然从图中来看max_lr越大, 那么 warmup 段的斜率越大 (还没有到达发散的临界点时)

- 2 可以推断: 在进入发散 lr 的领域之前,lr 越大越优秀

5.3 Problem : Batch size variations



(a) Batch size = 64 (b) Batch size = 128 (c) Batch size = 256

图 2: 不同 batch size 下的训练损失曲线（已平滑）

较小的 batch size 虽然损失曲线波动较大，但在训练初期收敛速度更快（相对的，按百分比计算）。

较大的 batch size 曲线更平滑（最后的部分），但达到相似的损失值可能需要更多的训练轮次。

较大的 batch size 可以减少梯度估计的方差，使损失曲线更加平稳。

增加 batch size 通常需要相应增大学习率, 调整学习率, 保证训练量 (total tokens processed) 相同的情况下，三种 batch size 最终都可以达到相似的损失值。

5.4 Problem：Generate text

贴一段 generate 出来的文本:

Prompt：hello my friend!

Reply：Can I play with you?" The cat smiled and said, "Yes, let's play together!" As they played, the cat and the dog found a big box. They didn't know what was inside. They decided to open it. Inside the box, they found a big cake. The cat and the dog were very surprised. They wanted to eat the cake, but they didn't know how to share. Just then, a little bird flew down and said, "I can help you open the box." The cat and the dog were happy. They opened the box, and there was a big cake inside. The cat

设置的参数:

$$\begin{aligned} temperature &= 0.8 \\ top\_k &= 10 \\ max\_new\_tokens &= 128 \end{aligned}$$

5.5 Problem：Remove RMSNorm and train

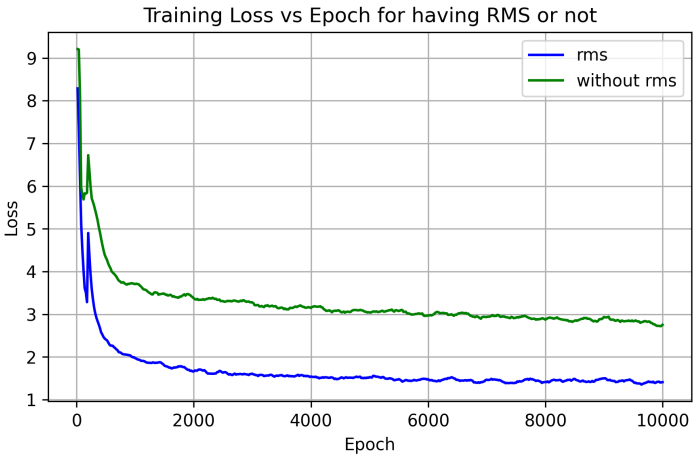


图 3: rms curve

没有 rms, 只能采用非常保守的 lr, 甚至需要让 cos schedule 在 800 轮就到达`min_lr`, 进而导致收敛缓慢

5.6 Problem : Implement post-norm and train

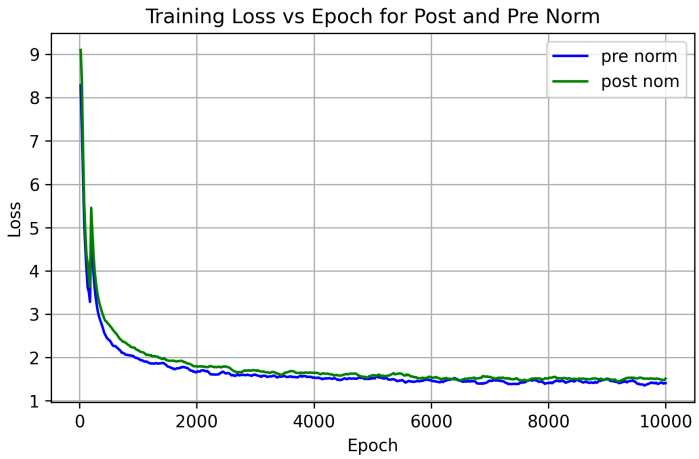


图 4: post norm curve

pre-norm 的版本下降速度更快, 收敛至更小的 loss

5.7 Problem : Implement NoPE

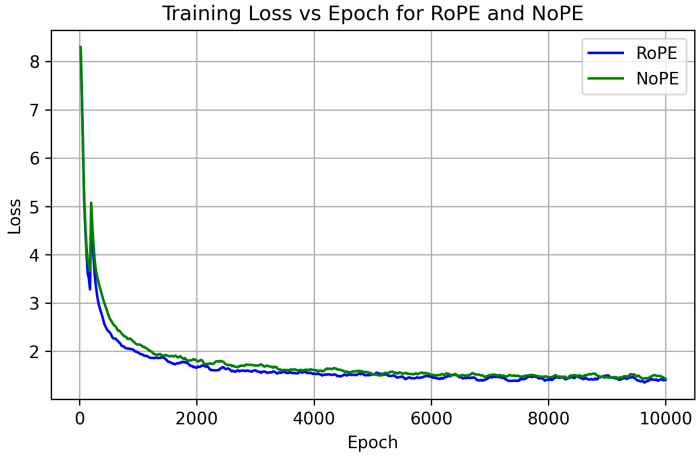


图 5: NoPE curve

RoPE 更好, 不过在 tiny story 的训练上, 这种差距并不明显, 但是再 open web text 上, 长程信息很重要时, 这种差距就会增大了。

5.8 Problem : SwiGLU vs. SiLU

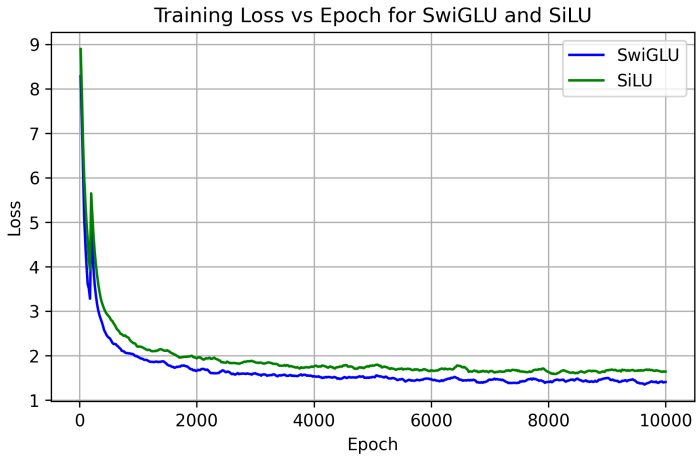


图 6: SiLU curve

很显然,SwiGLU 更好，下降略快，收敛值更低

5.9 Leaderboard

使用参数:

$$
\begin{aligned}
vocab_size &= 32000 \\
context_length &= 512 \\
d_model &= 512 \\
num_heads &= 16 \\
d_ff &= 2048 \\
num_layers &= 8
\end{aligned}
$$

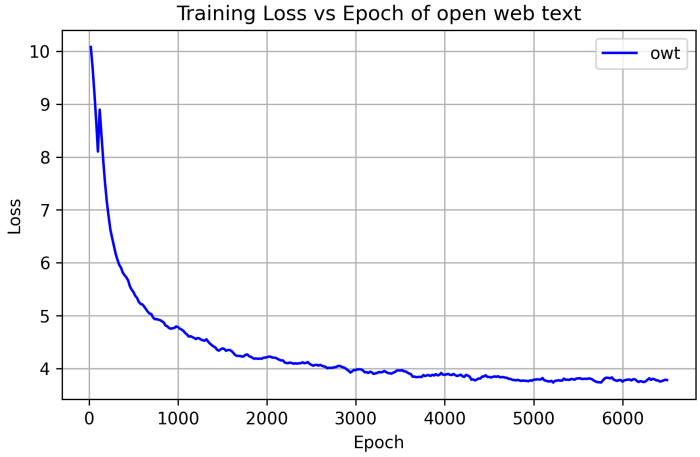


图 7: open web text curve

最后一轮 loss 为 3.72

一组输入输出：

Prompt：python

Reply：2.6 on Windows.This may be because the CTO is looking to get a new version of the kernel and we are looking to test it out with the CTO, but the latest version of Microsoft’s code doesn’t make it easy to test it.The reason we started this article? Can we make the CTO code?Microsoft has been working hard with the CTO to get a new CTO out of the way, so there is no need to worry about us getting a new CTO and getting a new CTO. A new CTO is going to be a way to