

第 0 周 · Day 2：优化与训练动力学

时长 7–8 小时：§1（梯度下降在做什么）2h · §2–§3（优化器）1.5h · §4（归一化）2h · §5–§7（初始化、调度、诊断）1h · §A–§C 1.5h

配套代码 `day2_checks.py`（纯 numpy）：最速方向与范数、最优步长与牛顿法、GD 的 $O(\kappa)$ vs 动量的 $O(\sqrt{\kappa})$ 、LayerNorm 雅可比的投影结构、Pre-LN/Post-LN 随深度的梯度缩放、三种初始化的 10 层对比、Adam 偏差修正、AdamW vs L2

§B 参考实现（`code/` 目录）：`d2_layernorm_autograd.py`、`d2_diagnostics.py`、`d2_optimizers_numpy.py`。先自己写，再对照；一键自检 `python code/run_selftests.py`，说明见 `code/README.md`。

§0 今天的主线

0.1 一个问题

Day 1 回答了"梯度怎么算"。今天回答：

有了梯度之后，怎么把参数更新到好的地方？

听起来只是一行 $\theta \leftarrow \theta - \eta g$ 。但这一行有四种独立的失败方式，现代训练的四大组件正好各治一种：

失败方式	组件	本文
梯度方向本身不好使（地形病态）	优化器	§1–§3
每层输出的尺度失控，连乘爆炸或消失	归一化	§4
出发点就错了，第一步就崩	初始化	§5
训练不同阶段该用不同步长	学习率调度	§6

0.2 贯穿全天的一个视角

今天的核心不是记住五六个优化器的公式，而是建立一个**统一视角**（§1.5）：

每一个优化器，都是在某一把"尺子"下做最速下降。

SGD 用欧氏尺子，牛顿法用 Hessian 当尺子，Adam 用一把对角尺子，符号下降用 L_∞ 尺子。有了这个视角，AdaGrad、RMSProp、Adam、甚至 2024 年的 Muon，都不再是一堆独立的技巧，而是同一个问题（"尺子该怎么选"）的不同回答。

归一化和初始化也能放进这个视角：它们不改尺子，而是**改地形本身**，让地形对任何尺子都更友好。

0.3 今天会用到的 Day 1 结论

- Day 1 §7.2：梯度是矩阵连乘，增益偏离 1 就指数爆炸或消失 → 归一化和初始化的动机
- Day 1 §3.1：梯度与参数同形状 → Adam 为什么吃 2 倍参数量的显存
- Day 1 §7.2：残差的恒等路径 → Pre-LN 为什么好训

0.4 和正课的关系

- **Lecture 1–2**: LayerNorm、RMSNorm、Pre-LN (§4)
- **Lecture 4**: AdamW 的显存、混合精度、ZeRO (§3、§A.5)
- **所有 Transformer 训练**: warmup + cosine、残差缩放初始化 (§5.3、§6)

§1 梯度下降到底在做什么

1.1 梯度为什么是"最陡方向": 取决于你用什么尺子量

动机。人人都说"梯度是下降最快的方向", 但很少有人追问: "**最快**"是相对什么说的? 这个追问是今天整个统一视角的起点。

推导。一阶泰勒展开: 沿方向 d 走一小步 ϵ ,

$$L(\theta + \epsilon d) \approx L(\theta) + \epsilon g^\top d$$

要找让 L 下降最多的方向, 就是在"长度为 1 的方向"里最小化 $g^\top d$ 。注意"长度为 1"需要一把尺子。用欧氏长度 $\|d\|_2 = 1$, 由柯西-施瓦茨不等式,

$$g^\top d \geq -\|g\|_2 \|d\|_2 = -\|g\|_2$$

等号当且仅当 $d = -g/\|g\|_2$ 时成立。所以在欧氏尺子下, 最陡方向是负梯度。

换一把尺子, 答案就变了。

- 用 L_∞ 尺子 ($\max_i |d_i| = 1$): 每个坐标可以独立取到 ± 1 , 最小化 $\sum_i g_i d_i$ 就是让每一项都取负, 于是 $d = -\text{sign}(g)$ 。这叫**符号下降**。
- 用由正定矩阵 P 定义的尺子 ($\sqrt{d^\top P d} = 1$): 令 $e = P^{1/2}d$, 则 $g^\top d = (P^{-1/2}g)^\top e$, $\|e\|_2 = 1$, 由上面的结论 $e \propto -P^{-1/2}g$, 换回来 $d \propto -P^{-1}g$ 。这叫**预条件梯度下降**。

`day2_checks.py` 在 40 万个随机方向里暴力搜索最优方向, 三种尺子下都和公式方向对上了 (余弦 0.990–0.998, 差距来自有限采样)。

意义。"梯度是最陡方向"这句话只在欧氏几何下成立。欧氏尺子是一个选择, 而且通常不是好的选择 (§1.3 会看到为什么)。所有优化器的差别, 本质上都是尺子的差别。§1.5 会把这件事讲完整。

1.2 步长该多大: 由曲率决定

动机。§1.1 只给了方向, 没说走多远。一阶展开里 ϵ 越大下降越多, 显然不对, 因为一阶近似只在局部成立。要知道走多远, 得看二阶。

推导。二阶泰勒展开, H 是 Hessian:

$$L(\theta - \eta g) \approx L(\theta) - \eta g^\top g + \frac{1}{2}\eta^2 g^\top H g$$

这是 η 的二次函数, 对 η 求导令其为 0:

$$-g^\top g + \eta g^\top H g = 0 \quad \Rightarrow \quad \boxed{\eta^* = \frac{g^\top g}{g^\top H g}}$$

`day2_checks.py` 用网格搜索验证了这个公式 (0.013170 vs 0.013166)。

直觉。 如果 g 恰好沿着 H 的某个特征向量 (特征值 λ)，那么 $\eta^* = 1/\lambda$ 。最优步长就是那个方向上曲率的倒数：地形越陡越弯，步子越小；越平越直，步子越大。

问题在于：不同方向的曲率可以差几个数量级，而梯度下降只有一个 η 。下一节看这会导致什么。

1.3 病态条件：一个能算到底的模型

用二次型分析，因为它能算到底，而且任何光滑损失在极小点附近都近似二次：

$$L(\theta) = \frac{1}{2}\theta^\top H\theta, \quad H = Q\Lambda Q^\top \text{ (对称正定)}$$

梯度 $\nabla L = H\theta$ ，梯度下降是 $\theta_{t+1} = (I - \eta H)\theta_t$ 。在特征基下 ($\tilde{\theta} = Q^\top \theta$) 各方向完全解耦：

$$\tilde{\theta}_{i,t} = (1 - \eta\lambda_i)^t \tilde{\theta}_{i,0}$$

收敛要求对每个 i 都有 $|1 - \eta\lambda_i| < 1$ ，即

$$0 < \eta < \frac{2}{\lambda_{\max}}$$

这就是那个要命的矛盾：

- 学习率上界被最大曲率 $\lambda_{\max}$ 卡死：超过它，最陡的方向就开始震荡发散
- 收敛速度被最小曲率 $\lambda_{\min}$ 拖住：最平的方向每步只挪一点点

取最优固定步长 $\eta = \frac{2}{\lambda_{\max} + \lambda_{\min}}$ ，最慢方向每步收缩 $\frac{\kappa-1}{\kappa+1}$ ，其中 $\kappa \equiv \lambda_{\max}/\lambda_{\min}$ 是条件数。所需步数正比于 κ 。[day2_checks.py](#) 实测：

κ	梯度下降步数 (误差降到 10^{-6})
10	66
100	609
1000	6,471
10000	62,586

κ 每乘 10，步数就乘 10。深度网络的条件数动辄 10^4 以上。

几何图像： 一个极其扁长的峡谷。梯度几乎垂直于峡谷轴线，于是你在陡峭的两壁之间来回横跳，沿谷底（真正要去的方向）几乎不动。

诊断句式： 训练不动 = 病态条件 + 学习率被最陡方向卡住。

1.4 牛顿法：理想答案，以及它为什么用不了

直觉。 §1.2 说最优步长是曲率的倒数，§1.3 说麻烦在于不同方向曲率不同。那给每个方向各配一个步长，不就好了？

推导。 直接最小化整个二阶模型 $g^\top \Delta + \frac{1}{2}\Delta^\top H\Delta$ ，对 Δ 求导令其为 0：

$$g + H\Delta = 0 \quad \Rightarrow \quad \Delta = -H^{-1}g$$

在二次型上它一步到位（`day2_checks.py`：一步之后 $\|\theta\| = 1.1 \times 10^{-14}$ ）。在特征基下看得最清楚： H^{-1} 把第 i 个方向的步长恰好设为 $1/\lambda_i$ ，所有方向同时被"拉平"，条件数变成 1。

和 §1.1 连起来：牛顿法就是用 $P = H$ 这把尺子做最速下降。用地形自己的曲率去量长度，陡的方向"尺子上的一格"很短，平的方向很长，病态条件就消失了。

为什么用不了？三个理由，每一个都致命：

- 1. 存不下：70 亿参数的 Hessian 有 4.9×10^{19} 个元素。
- 2. 求逆太贵： $O(n^3)$ 。
- 3. 非凸时方向是错的：深度网络的 Hessian 有负特征值（鞍点）， $-H^{-1}g$ 在负曲率方向上会往上爬，被鞍点吸过去。

意义。牛顿法给出了一个"理想"，而后面所有实用优化器都可以理解为：用便宜的办法近似 H 这把尺子。对角近似（AdaGrad、Adam）、分块近似（K-FAC、Shampoo）、低秩近似（L-BFGS），都是在存储、计算和效果之间找平衡。

1.5 统一视角：每个优化器都是在选一把尺子

把 §1.1 和 §1.4 合起来：

$$\Delta\theta = -\eta P^{-1}g$$

选不同的 P ，就得到不同的优化器：

优化器	尺子 P	代价
SGD	I （欧氏）	无，但对病态条件毫无办法
牛顿法	H	存不下、算不起、非凸时方向错
AdaGrad / Adam	对角阵，由梯度历史估计	$O(n)$ ，只能纠正坐标轴方向上的尺度差异
符号下降	L_∞ 范数	无，每个坐标只看符号
Shampoo / Muon	按矩阵结构（行、列）构造	几倍于 Adam 的计算，能纠正相关方向上的差异

最后一行是近年的方向。以 Muon（2024）为例，它把参数看成矩阵而不是一串数字，用矩阵的谱范数当尺子。在谱范数下，§1.1 那个"最小化 $\text{tr}(G^\top D)$ s.t. $\|D\| \leq 1$ "的解是 $D = -UV^\top$ （ $G = U\Sigma V^\top$ 是梯度的 SVD），也就是把梯度矩阵的所有奇异值拉成 1。Muon 实际做的是用几步 Newton-Schulz 迭代近似地把（带动量的）更新矩阵正交化，因为每步都做精确 SVD 太贵。`day2_checks.py` 验证了这个解的取值恰好等于 $-\|G\|_*$ （核范数），20 万个随机方向都没能超过它。

这个视角的价值：以后看到任何新优化器，先问一句"它用的是什么尺子"，就能迅速定位它在解决什么问题、会付出什么代价。

1.6 随机性：mini-batch 梯度的方差

真实训练用 mini-batch 梯度，它是全量梯度的无偏估计，但有方差：

$$\mathbb{E}[g_B] = g, \qquad \text{Var}[g_B] \propto \frac{1}{B}$$

方差有两面：靠近极小点时噪声让参数在附近抖而不收敛（所以需要学习率衰减，§6.2）；但噪声也帮助逃离鞍点和尖锐极小点，普遍认为这是 SGD 隐式正则效果的来源之一。

这也解释了 **batch size 和学习率的关系**： B 增大、噪声变小，可以用更大的学习率。SGD 下"噪声尺度"约为 η/B ，要保持它不变就是**线性缩放** $\eta \propto B$ ；Adam 的更新已经被 $\sqrt{\hat{v}}$ 归一化过，经验上更接近 $\eta \propto \sqrt{B}$ 。这是经验规律，大 batch 下两者都会失效。

§2 优化器的演化链

每一步只解决上一步的一个具体问题。带着 §1.5 的"尺子"视角看，这条链会非常清楚。

2.1 SGD

$$\theta_{t+1} = \theta_t - \eta g_t$$

尺子是 I 。§1.3 的病态条件问题原封不动。

2.2 Momentum：不换尺子，也能把 κ 降成 $\sqrt{\kappa}$

动机。换尺子要估计曲率，贵。有没有不估曲率也能对付峡谷的办法？

直觉：重球。想象一个有质量的球滚进峡谷。在两壁之间，它每次撞壁都被反弹，**来回的冲量互相抵消**；沿谷底，每一步的推力方向都相同，**冲量不断累积**。惯性自动压掉了横向震荡，放大了纵向前进。

$$b_t = \mu b_{t-1} + g_t, \quad \theta_{t+1} = \theta_t - \eta b_t$$

（这是 PyTorch `SGD(momentum=μ)` 的实际形式。注意 g_t 前**没有** $(1 - \mu)$ ，和 EMA 形式差一个缩放。）

推导一：有效步长的放大。假设梯度恒为 g ，稳态时 $b = \mu b + g$ ，解得 $b = \frac{g}{1-\mu}$ ，于是

$$\text{有效学习率} = \frac{\eta}{1 - \mu}$$

$\mu = 0.9$ 时是 **10 倍**， $\mu = 0.99$ 是 **100 倍**。**开动量时必须相应调小学习率**，否则直接发散，这是新手最常踩的坑之一。

推导二：为什么是 $\sqrt{\kappa}$ (Polyak 1964)。这一段回答"动量到底强在哪"，不只是"平滑"那么简单。

在二次型的第 i 个特征方向上，重球法的迭代是（写成 x 以免下标太多）：

$$x_{k+1} = (1 + \beta - \alpha\lambda) x_k - \beta x_{k-1}$$

这是一个**二阶线性递推**（梯度下降是一阶的）。它的特征方程是

$$r^2 - (1 + \beta - \alpha\lambda)r + \beta = 0$$

关键观察：当判别式 ≤ 0 ，两个根是一对共轭复数，模长 $|r| = \sqrt{\beta}$ ，与 λ **无关**。也就是说，只要所有特征方向的判别式都 ≤ 0 ，**所有方向以同一个速率 $\sqrt{\beta}$ 收敛**，曲率差异被抹平了。

判别式 ≤ 0 等价于 $|1 + \beta - \alpha\lambda| \leq 2\sqrt{\beta}$ ，即

$$(1 - \sqrt{\beta})^2 \leq \alpha\lambda \leq (1 + \sqrt{\beta})^2$$

要对所有 $\lambda \in [\lambda_{\min}, \lambda_{\max}]$ 成立，两端相除：

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}} \leq \left(\frac{1 + \sqrt{\beta}}{1 - \sqrt{\beta}} \right)^2 \Rightarrow \sqrt{\beta} \geq \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}$$

取等号得到最快速率 $\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$ ，所需步数正比于 $\sqrt{\kappa}$ （对应的 $\alpha = 4/(\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}})^2$ ）。

day2_checks.py 实测：

κ	GD 步数	重球步数	倍数
10	66	26	2.5
100	609	80	7.6
1000	6,471	303	21.4
10000	62,586	933	67.1

κ 每乘 10，GD 步数乘 10，重球只乘约 $\sqrt{10} \approx 3.2$ 。

意义。 动量不是"平滑一下噪声"的小技巧，它改变了收敛的阶：从 $O(\kappa)$ 到 $O(\sqrt{\kappa})$ ，而且一分钱曲率信息都没用。这也是为什么后面的 Adam 保留了动量这一项。（上面是凸二次型上的理论；深度网络非凸、有噪声，不会严格满足，但"动量对病态地形特别有效"这个结论在实践中非常稳健。）

2.3 Nesterov：先看一步再走

$$b_t = \mu b_{t-1} + \nabla L(\theta_t - \eta \mu b_{t-1})$$

在"动量将要把我带到的位置"求梯度，一种前瞻修正。凸优化里理论更好，深度学习里提升有限。知道它存在即可。

2.4 AdaGrad：用梯度历史估一把对角尺子

动机。 回到 §1.5：想要一把比 I 好、又比 H 便宜的尺子。**对角阵**是最便宜的选择，只需要 n 个数。问题是：对角线上该填什么？

直觉。 真正的曲率算不起，但梯度是免费的。一个坐标的梯度一直很大，说明这个方向上损失变化剧烈，步子该小；梯度一直很小或很稀疏（比如 embedding 里的低频词），步子该大。于是用梯度的历史幅度来当尺子：

$$v_t = v_{t-1} + g_t^2, \quad \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} g_t \quad (\text{逐元素})$$

也就是 $P = \text{diag}(\sqrt{v_t})$ 。开根号的严格理由来自在线凸优化的 regret 分析（Duchi et al. 2011），直观理解是：把每个坐标的梯度除以它的典型幅度，让所有坐标的步长回到同一量级。

致命缺陷： v_t 单调累加，只增不减。训练久了 $\sqrt{v_t}$ 无限增大，学习率单调衰减到 0，训练提前停死。凸问题里这是个特性（保证收敛），深度学习里是个 bug。

2.5 RMSProp：让尺子会"遗忘"

一行修正，把累加改成指数移动平均：

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

v_t 只反映"最近一段时间"的梯度幅度，不会无限增长。深度学习的地形一直在变，尺子也该跟着变。

2.6 Adam: 动量 + 自适应尺子 + 偏差修正

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t && \text{一阶矩: 方向 (§2.2 的动量)} \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 && \text{二阶矩: 尺子 (§2.5)} \\ \hat{m}_t &= \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} && \text{偏差修正} \\ \theta_{t+1} &= \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \end{aligned}$$

默认 $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ 。

偏差修正从哪来

动机。 $m_0 = 0$ 是一个人为的、错误的初值，它会把早期的估计拉向零。拉了多少？

推导。 把递推展开：

$$m_t = (1 - \beta_1) \sum_{i=1}^t \beta_1^{t-i} g_i$$

假设各步梯度同分布、期望为 $\bar{g}$ ，用等比数列求和：

$$\mathbb{E}[m_t] = (1 - \beta_1) \bar{g} \sum_{i=1}^t \beta_1^{t-i} = (1 - \beta_1) \bar{g} \cdot \frac{1 - \beta_1^t}{1 - \beta_1} = \bar{g} (1 - \beta_1^t)$$

偏差因子正是 $(1 - \beta_1^t)$ ，除掉它就无偏。 $t = 1$ 时估计只有真值的 10%； $\beta_2 = 0.999$ 那一路更是只有 0.1%。偏差修正不是可有可无的小修补，前几百步没它就是错的。（`day2_checks.py`：修正后每个 t 都精确等于真值。）

Adam 的第一步

$t = 1$ 时 $\hat{m}_1 = g_1$, $\hat{v}_1 = g_1^2$ ，于是

$$\Delta\theta = -\eta \frac{g_1}{|g_1| + \epsilon} \approx \mp \eta$$

无论梯度多大多小，第一步长度都约等于 η 。`day2_checks.py` 把梯度从 10^{-2} 调到 10^3 ，第一步都是 η ；只有当 $|g|$ 小到和 ϵ 可比（如 10^{-6} ）时才略小于 η 。

这个"步长与梯度尺度无关"的性质，既是 Adam 好用的原因（不用手调各层学习率），也是它早期不稳的原因（一上来就是满幅、方向近乎随机的大步）。§6.1 的 **warmup** 就是来治这个的。

Adam 的 $\sqrt{\hat{v}}$ 到底是不是曲率？（诚实版）

常见说法是"Adam 用梯度二阶矩近似 Hessian 对角"。这个说法只对了一半，要说清楚：

- 对于对数似然型损失，梯度外积的期望 $\mathbb{E}[gg^\top]$ 是 **Fisher 信息矩阵**，在最优点附近它近似 Hessian（Gauss-Newton 近似）。所以 v 确实和曲率有关。

- 但 Adam 用的是 $\sqrt{v}$ ，不是 v 。真正的牛顿法对角近似应该除以 v ，Adam 除以的是它的平方根。所以 Adam 不是对角牛顿法。
- 更好的理解方式：令 $\beta_1 = \beta_2 = 0$ ，Adam 退化为 $g/|g| = \text{sign}(g)$ ，正是 §1.1 的 L_∞ 最速下降。Adam 最好被理解成“带平滑的符号下降”： β_1 平滑方向， β_2 平滑尺度。

这个理解能解释很多现象：为什么 Adam 对梯度尺度不敏感（符号不关心幅度）、为什么它在不同层之间自动平衡步长、以及为什么纯符号方法（如 Lion, 2023）能和 Adam 打平。

2.7 AdamW：把 weight decay 从尺子里解放出来

面试高频题，而且很多人只会背结论。

动机。 用 L2 正则的方式做 weight decay：损失加上 $\frac{\lambda}{2} \|\theta\|^2$ ，梯度变成 $g' = g + \lambda\theta$ 。这个 g' 会被送进 Adam 的尺子，实际衰减量变成

$$\eta \cdot \frac{\lambda\theta}{\sqrt{\hat{v}} + \epsilon}$$

衰减被 $\sqrt{\hat{v}}$ 除了。梯度大的参数 $\sqrt{\hat{v}}$ 大，几乎不被衰减；梯度小的参数反而被过度衰减。这违背了 weight decay 的初衷：对所有参数一视同仁地往 0 收缩。

直觉。 问题出在 weight decay 和梯度走了同一个通道。把它从梯度通道里拿出来，不经过尺子，直接作用在参数上：

$$\theta_{t+1} = \theta_t - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda\theta_t \right)$$

day2_checks.py 对比 ($\lambda = 0.1, \eta = 0.1, \theta = 1$) :

	$ g = 0.01$	$ g = 1$	$ g = 10$
Adam + L2 实际衰减	1.000	0.010	0.001
AdamW 衰减	0.010	0.010	0.010

Adam+L2 在两端之间差 **1000 倍**，AdamW 完全一致。

意义。 对 SGD 来说 L2 和 weight decay 是一回事（尺子是 I ，除了等于没除）；只有在自适应尺子下两者才分道扬镳。这就是为什么 `torch.optim.Adam(weight_decay=...)` 和 `AdamW(weight_decay=...)` 不是同一件事，以及为什么现代 Transformer 训练全部用 AdamW。

实践补充： weight decay 通常不施加在 bias 和 LayerNorm 的 γ, β 上。这些参数不是“权重”，收缩它们没有正则意义反而有害。实现方式见 §A.1。

2.8 汇总

优化器	尺子 P	解决了什么	留下了什么
SGD	I	—	病态条件
+ Momentum	I	$\kappa \rightarrow \sqrt{\kappa}$ ，治震荡	仍是全局同一尺度
AdaGrad	$\text{diag} \sqrt{\sum g^2}$	逐坐标自适应	学习率单调衰减到 0

优化器	尺子 P	解决了什么	留下了什么
RMSProp	$\text{diag} \sqrt{\text{EMA}(g^2)}$	尺子会遗忘	没有动量
Adam	同上 + 偏差修正	合体, $\approx$ 带平滑的符号下降	weight decay 被尺子污染
AdamW	同上	解耦 weight decay	当前标准

§3 优化器的显存账

接 Day 1 §9。AdamW 为每个参数维护 m 和 v ，各自与参数同形状（Day 1 §3.1），fp32 各 4 字节：

$$\text{优化器状态} = 8 \text{ 字节/参数}$$

这是 Day 1 §9.1 那张 16 字节表里最大的一块。7B 模型光优化器状态就是 **56 GB**。由此理解两个工程方向：

- **8-bit 优化器** (**bitsandbytes**)： m, v 量化到 8-bit, $8 \rightarrow 2$ 字节/参数，7B 省 42 GB。QLoRA 的 paged optimizer 属于这一类。
- **ZeRO Stage 1**：把优化器状态切到 N 张卡上，每张只存 $1/N$ 。ZeRO 三阶段里最先做、收益最大的一步，因为它本来就是最大头。

SGD+Momentum 只要 4 字节/参数（一个动量），是 AdamW 的一半；符号类优化器（Lion）也只存一个动量。超大规模训练里偶尔有人为这个理由考虑它们，省下的显存是实打实的。

§4 归一化

4.1 它到底在解决什么

动机。原始 BatchNorm 论文给的理由是“减少 internal covariate shift”（层输入分布随训练漂移）。**这个解释后来被实验否定了**：Santurkar 等人（2018）人为往归一化后注入分布漂移，训练照样又快又稳。

目前广泛接受的解释有两条，都能放进今天和昨天的框架里：

1. **改善条件数**：归一化让损失曲面更平滑，也就是 §1.3 里的 $\lambda_{\max}$ 变小，**可用的学习率上界变大**。用 §0.2 的话说，它不换尺子，而是**改地形**，让欧氏尺子也够用。
2. **控制连乘增益**：Day 1 §7.2 说梯度是矩阵连乘，增益偏离 1 就指数爆炸或消失。归一化把每层输出的尺度钉在 $O(1)$ ，等于给连乘装了限幅器。

第二条是理解 Transformer 的关键：100 层网络没有归一化根本训不起来，不是因为分布漂移，而是因为连乘。

4.2 BatchNorm，以及它为什么不适合 NLP

BatchNorm 在 **batch 维**上统计：对每个特征通道，算这一批样本的均值方差。

三个问题，任何一个在 NLP 里都是致命的：

1. **依赖 batch size**：batch 小时统计量噪声大，batch=1 时根本没法算。
2. **训练与推理不一致**：推理时没有 batch，只能用训练期间的滑动平均。多了一份状态，也多了一个 bug 温床（Day 1 §A.3 坑 2）。

3. **变长序列直接崩**：一个 batch 里句子长短不一，padding 会污染统计量；而且同一个位置在不同句子里语义完全不同，凭什么放在一起统计？

4.3 LayerNorm：在特征维上归一化

直觉。BatchNorm 的三个问题全都来自"跨样本统计"。那就不跨样本，只在单个样本自己的特征维内部统计。

$$\mu = \frac{1}{d} \sum_i x_i, \quad v = \frac{1}{d} \sum_i (x_i - \mu)^2, \quad \sigma = \sqrt{v + \epsilon}, \quad \hat{x}_i = \frac{x_i - \mu}{\sigma}, \quad y_i = \gamma_i \hat{x}_i + \beta_i$$

与 batch 无关、训练推理完全一致、变长序列毫无影响。**NLP 用 LayerNorm 不是习惯，是被逼的。**

反向推导：三条路径，索引级

动机。这里是 Day 1 §7.1 多路径规则的一次正式演练。 x_j 影响 $\hat{x}_i$ 有**三条路**：

1. 直接影响（仅当 $i = j$ ）
2. 通过 μ （ μ 依赖所有 x_j ）
3. 通过 σ （同上）

三条路，所以会有三项。下面把它们一项一项推出来。

零件一： $\frac{\partial \mu}{\partial x_j} = \frac{1}{d}$

零件二： v 对 x_j 的导数。对 $v = \frac{1}{d} \sum_k (x_k - \mu)^2$ 用链式法则，注意 μ 也依赖 x_j ：

$$\frac{\partial v}{\partial x_j} = \frac{2}{d} \sum_k (x_k - \mu) \left(\delta_{kj} - \frac{1}{d} \right) = \frac{2}{d} (x_j - \mu) - \underbrace{\frac{2}{d^2} \sum_k (x_k - \mu)}_{=0} = \frac{2}{d} (x_j - \mu)$$

中心化之后的和为 0，第二项直接消失。这一步经常被跳过，它是整个推导能化简的关键。

零件三： $\frac{\partial \sigma}{\partial x_j} = \frac{1}{2\sigma} \frac{\partial v}{\partial x_j} = \frac{x_j - \mu}{d\sigma}$

组装（商法则， $\hat{x}_i = (x_i - \mu)/\sigma$ ）：

$$\frac{\partial \hat{x}_i}{\partial x_j} = \frac{(\delta_{ij} - \frac{1}{d})\sigma - (x_i - \mu)\frac{\partial \sigma}{\partial x_j}}{\sigma^2} = \frac{1}{\sigma} \left(\delta_{ij} - \frac{1}{d} \right) - \frac{(x_i - \mu)(x_j - \mu)}{d\sigma^3}$$

用 $\hat{x}$ 改写，三项清清楚楚：

$$\boxed{\frac{\partial \hat{x}_i}{\partial x_j} = \frac{1}{\sigma} \left(\underbrace{\delta_{ij}}_{\text{直接}} - \underbrace{\frac{1}{d}}_{\text{经 } \mu} - \underbrace{\frac{\hat{x}_i \hat{x}_j}{d}}_{\text{经 } \sigma} \right)}$$

收缩上游梯度：记 $g = \bar{y} \odot \gamma$ （对 $\hat{x}$ 的梯度），

$$\frac{\partial L}{\partial x_j} = \sum_i g_i \frac{\partial \hat{x}_i}{\partial x_j} = \frac{1}{\sigma} \left[g_j - \bar{g} - \hat{x}_j \overline{g \odot \hat{x}} \right]$$

($\bar{\cdot}$ 表示在特征维上取均值。) 参数的梯度简单得多, γ, β 被所有样本和位置共享, 所以求和 (Day 1 §6.3) :

$$\frac{\partial L}{\partial \gamma} = \sum \bar{y} \odot \hat{x}, \quad \frac{\partial L}{\partial \beta} = \sum \bar{y}$$

`day2_checks.py` 逐项验证, 误差 10^{-10} 。

几何意义：雅可比是一个投影

把上面的雅可比写成矩阵：

$$J = \frac{1}{\sigma} \left(I - \frac{\mathbf{1}\mathbf{1}^\top}{d} - \frac{\hat{x}\hat{x}^\top}{d} \right)$$

看括号里的东西：

- $\mathbf{1}^\top \hat{x} = \sum_i \hat{x}_i = 0$, 所以 $\mathbf{1}$ 和 $\hat{x}$ 正交
- $\|\mathbf{1}\|^2 = d$, $\|\hat{x}\|^2 = d \cdot \frac{v}{v+\epsilon} \approx d$

所以 $\frac{\mathbf{1}\mathbf{1}^\top}{d}$ 和 $\frac{\hat{x}\hat{x}^\top}{d}$ 分别是到 $\mathbf{1}$ 方向和 $\hat{x}$ 方向的正交投影, I 减去它们就是到这两个方向的正交补空间的投影。

`day2_checks.py` 验证了 σJ 近似幂等、 $\sigma J\mathbf{1} = 0$ 、 $\sigma J\hat{x} \approx 0$ 。

意义。 这不是巧合, 背后有一条普遍规律：

输出对哪个方向的扰动不敏感, 梯度在那个方向上就为零。

LayerNorm 的输出对"整体平移输入" (沿 $\mathbf{1}$) 和"整体缩放输入" (沿 $x - \mu$, 也就是 $\hat{x}$ 方向) 都不变。所以反向时, 这两个方向上的梯度分量被直接丢掉。

还有一个后面马上要用的推论: J 前面有个 $1/\sigma$ 。输入的尺度越大, 穿过 LayerNorm 的梯度就越小。§4.5 会看到, 这正是 Pre-LN 在深层"变温和"的原因。

4.4 RMSNorm: 只管缩放, 不管平移

$$r = \sqrt{\frac{1}{d} \sum_i x_i^2 + \epsilon}, \quad y_i = \gamma_i \frac{x_i}{r}$$

动机。 LayerNorm 做了两件事: 重新中心化 (减 μ) 和重新缩放 (除 σ)。Zhang & Sennrich (2019) 的问题是: **两件事都必要吗?** 他们的实验表明好处主要来自缩放, 中心化可以砍掉。

推导。 只剩**两条路径** (直接、经 r) , 同样的方法推出:

$$\frac{\partial L}{\partial x} = \frac{1}{r} \left[g - x \cdot \frac{\overline{g \odot x}}{r^2} \right], \quad J = \frac{1}{r} \left(I - \frac{xx^\top}{dr^2} \right)$$

对照 §4.3: **少掉的正是 $\mathbf{1}\mathbf{1}^\top/d$ 那一项**, 也就是"平移不变"那一条。RMSNorm 的雅可比只投影掉缩放方向。(已数值验证。)

意义与代价。 少一次求均值、少一次减法、反向少一项, 实测快 7-15%, 质量无明显损失。**Llama、Qwen、Mistral、Gemma** 等现代模型全部采用 **RMSNorm**, Lecture 2 看架构对比时到处都是它。和 Day 4 的 GRU 简化 LSTM 是同一种方法论: **先问每个组件是否必要, 砍掉不必要的。**

4.5 Pre-LN vs Post-LN：为什么现代模型都用前者

Lecture 1-2 会直接考。

Post-LN (2017 原版) : $x_{l+1} = \text{LN}(x_l + f_l(x_l))$ **Pre-LN** (现代) : $x_{l+1} = x_l + f_l(\text{LN}(x_l))$

看起来只是括号位置不同。用 Day 1 §7.2 的残差分析和 §4.3 的 $1/\sigma$ ，差别就清楚了。

Pre-LN 的推导。 把 L 层展开，残差流是所有分支输出的累加：

$$x_L = x_l + \sum_{k=l}^{L-1} f_k(\text{LN}(x_k)) \quad \Rightarrow \quad \frac{\partial x_L}{\partial x_l} = I + \sum_k (\cdots)$$

那个 I 是一条从第 L 层直通第 l 层、一次 **LayerNorm** 都不经过的路径。梯度高速公路完全畅通。

但累加有一个副作用：每个分支输出 $O(1)$ ， L 个近似独立的 $O(1)$ 向量相加，范数按 $\sqrt{L}$ 增长。现代 Pre-LN 模型在输出头之前还有一个**末端 LayerNorm**，而由 §4.3，穿过它的梯度要乘 $1/\sigma \propto 1/\sqrt{L}$ 。于是所有参数的梯度都被缩小了 $\sqrt{L}$ 倍：网络越深，Pre-LN 越“温和”。

Post-LN 的情况。 每层都重新归一化，残差流范数被钉死在 $\sqrt{d}$ ，靠近输出的参数梯度不会随深度缩小。

`day2_checks.py` 在随机初始化的 $d = 64$ 残差网络上实测（8 个随机种子平均）：

层数 L	Pre-LN 残差流范数	Pre-LN 末层参数梯度	Post-LN 残差流范数	Post-LN 末层参数梯度
6	15.2	41.2	8.0	63.3
24	26.2	23.7	8.0	64.1
96	46.8	13.3	8.0	64.2
384	95.9	6.5	8.0	65.3

层数每乘 4，Pre-LN 的残差流范数翻倍 ($\propto \sqrt{L}$)、梯度减半 ($\propto 1/\sqrt{L}$)；Post-LN 两者都不变。

意义：为什么 Post-LN 离不开 warmup。 用同一个学习率，深的 Post-LN 在初始化时靠近输出的参数梯度比 Pre-LN 大 $\sqrt{L}$ 量级（384 层时大 10 倍），第一批更新步子太大，训练不稳。Xiong 等人（2020）从理论上分析了这个机制，并显示 Pre-LN 可以去掉 warmup 或大幅缩短它。（另有一条互补的解释：Liu 等人（2020）认为 Post-LN 对残差分支的依赖过强，会放大参数扰动对输出的影响。两种机制都存在。）

Pre-LN 的代价：表里左边那列同时说明了一件坏事。残差流按 $\sqrt{L}$ 增长，每个新分支只贡献 $O(1)$ ，越深的层对残差流的相对影响越小，深层的边际贡献递减。这是 Pre-LN 的已知问题，也是 DeepNorm、Sandwich-LN、以及 §5.3 残差分支缩放这类改进的动机。

实践结论：绝大多数 LLM 用 Pre-LN（通常是 RMSNorm）加末端归一化。能稳定训起来，比最后那一点质量更重要。

§5 初始化：出发点决定生死

5.1 二阶矩传播的推导

动机。 Day 1 §7.2 说梯度是连乘。连乘里每一层的增益，在训练开始时完全由初始化决定。初始化的目标就是让每层的增益恰好是 1，前向信号和反向梯度都不放大也不缩小。

推导。 考虑一层 $z_i = \sum_{j=1}^n W_{ij} a_j$ (n 是 fan_in, 忽略偏置)。 W_{ij} 独立同分布、均值 0、方差 σ_W^2 ，且与 a 独立。

$$\mathbb{E}[z_i^2] = \sum_{j,k} \mathbb{E}[W_{ij} W_{ik}] \mathbb{E}[a_j a_k] = \sum_j \mathbb{E}[W_{ij}^2] \mathbb{E}[a_j^2] = n \sigma_W^2 \mathbb{E}[a^2]$$

$j \neq k$ 的交叉项因为 $\mathbb{E}[W_{ij} W_{ik}] = 0$ 全部消失。

注意这里出现的是 $\mathbb{E}[a^2]$ (二阶矩)，不是 $\text{Var}[a]$ 。很多教材写成方差，这在 a 均值为 0 时没区别，但 **ReLU** 的输出全是非负数，均值不为 0，此时必须跟踪二阶矩。(day2_checks.py: $\text{Var}[\text{ReLU}(z)] = 0.99$ ，而 $\frac{1}{2} \text{Var}[z] = 1.45$ ，两者不等。)

ReLU 那一步： 若 z 关于 0 对称，

$$\mathbb{E}[\text{ReLU}(z)^2] = \int_0^\infty z^2 p(z) dz = \frac{1}{2} \int_{-\infty}^\infty z^2 p(z) dz = \frac{1}{2} \mathbb{E}[z^2]$$

(对称分布的正半边恰好贡献一半。已数值验证：1.4459 vs 1.4458。)

合起来，相邻两层激活前的二阶矩满足

$$\mathbb{E}[z^{(l)2}] = n \sigma_W^2 \cdot \frac{1}{2} \mathbb{E}[z^{(l-1)2}]$$

要让增益为 1:

$$\boxed{\sigma_W^2 = \frac{2}{n}} \quad (\text{He 初始化, ReLU 家族})$$

那个 2 就是用来补偿 ReLU 砍掉的一半。对线性或 tanh (在 0 附近 $a \approx z$ ，均值为 0) 没有这个一半，就是 $\sigma_W^2 = 1/n$ (LeCun / Xavier 的 fan_in 版本)。

反向也有一份镜像。 反向时 $\bar{a} = W^\top \bar{z}$ ，求和是沿输出维进行的，同样的推导给出 $\sigma_W^2 = 2/n_{\text{out}}$ 。前向要 fan_in、反向要 fan_out，两者不能同时满足 (除非方阵)。Xavier 的 $\frac{2}{n_{\text{in}} + n_{\text{out}}}$ 就是两者的折中；PyTorch 的 `mode='fan_in'/'fan_out'` 让你选保哪一边 (§A.7)。

5.2 数值演示

10 层 ReLU 网络 (day2_checks.py)，看"第 1 层之后每层乘多少":

初始化	每层增益 (理论)	第 10 层 / 第 1 层 (实测)	理论 $(\cdot)^9$
He, $\mathcal{N}(0, 2/n)$	1	1.15	1
Xavier, $\mathcal{N}(0, 1/n)$	1/2	1.8×10^{-3}	2.0×10^{-3}
PyTorch nn.Linear 默认, $\mathcal{U}(\pm 1/\sqrt{n})$	1/6	1.2×10^{-7}	9.9×10^{-8}

(实测和理论有 $\pm 20\%$ 左右的偏差，是有限宽度 $n = 512$ 下每层增益的随机涨落连乘 9 次放大的结果，数量级对上即可。)

第三行值得专门记住。`nn.Linear` 的默认初始化是 `kaiming_uniform(a=√5)`，算出来是 $\mathcal{U}(\pm 1/\sqrt{n})$ ，方差 $1/(3n)$ ，只有 He 的 $1/6$ 。这个默认值主要出于历史兼容，并非为深 ReLU 网络设计。浅网络无所谓，深网络必须自己初始化，否则 10 层就把信号压到 10^{-7} 。

这是一个“一个常数因子写错就训不起来”的例子，也说明初始化不是玄学，是可以推导的。

5.3 Transformer 的残差分支缩放

动机。 §4.5 看到 Pre-LN 的残差流是分支输出的累加，范数按 $\sqrt{L}$ 增长。能不能在初始化时就把它摁住？

推导。 L 层 Transformer 每层有两个残差分支（注意力、FFN），共 $2L$ 个。若各分支输出近似独立、二阶矩都是 s^2 ，累加后

$$\mathbb{E}[\|x_L\|^2] \approx \|x_0\|^2 + 2L \cdot s^2$$

要让它不随 L 增长，就把每个分支的输出投影按 $1/\sqrt{2L}$ 缩放，使 $s^2 \propto 1/(2L)$ 。

这是 GPT-2 起的标准做法。第 1 周你手写 mini-GPT 时，如果层数一多就训练不稳，八成是漏了这个缩放。

§6 学习率调度

6.1 Warmup：为什么 Transformer 几乎必须要

两个独立原因，前面都埋好了伏笔。

原因一（优化器侧，§2.6）： 训练初期 t 很小， $\hat{v}_t$ 是用极少的样本估出来的，方差极大，自适应步长 $\hat{m}/\sqrt{\hat{v}}$ 跟着剧烈抖动。§2.6 算过 $t = 1$ 时步长恰好是 η 、方向却近乎随机，一上来就是满幅乱走。warmup 在 $\hat{v}$ 稳定之前把步长压住。（RAdam 论文把这个机制形式化了。）

原因二（架构侧，§4.5）： 深的 Post-LN 在初始化时靠近输出的梯度比 Pre-LN 大 $\sqrt{L}$ 量级，不 warmup 就发散。Pre-LN 好得多，但大模型训练仍普遍保留 warmup 当保险。

典型配置：前 1–5% 的步数线性 warmup ($0 \rightarrow$ 峰值)。

6.2 Cosine decay

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos \frac{\pi t}{T} \right)$$

直觉。 训练前期要探索，步子大；后期要在极小点附近精细收敛，还要压住 §1.6 的梯度噪声，步子小。cosine 前期衰减慢、后期衰减快，正好符合这个节奏。经验上比线性和阶梯衰减都稳。

陷阱： cosine 的形状绑死在总步数 T 上，中途想延长训练，曲线就错位了。这是 WSD (warmup-stable-decay) 这类调度受欢迎的原因之一：中间是恒定段，可以随时延长，最后再衰减。

6.3 现代 LLM 的标准配方

优化器 AdamW, $\beta_1=0.9$, $\beta_2=0.95$ (大模型常用 0.95 而非 0.999), $wd=0.1$
 调度 linear warmup (1-5% steps) $\rightarrow$ cosine decay 到峰值的 10%
 裁剪 clip_grad_norm = 1.0
 精度 bf16 混合精度 (Day 1 §10.2: 不需要 loss scaling)
 归一化 RMSNorm + Pre-LN + 末端归一化
 初始化 正态 std=0.02, 残差输出投影再乘 $1/\sqrt{2L}$
 wd 排除 bias 和 norm 层的 γ/β 不做 weight decay

现在你能逐条说出每一项为什么长这样。 $\beta_2 = 0.95$ 是让二阶矩估计更"近视", 对大模型训练中偶发的梯度尖峰反应更快, 减少不稳定。这是今天最值钱的产出。

§7 五个诊断场景

训练出问题, 按 §1-§6 的框架排查。当真题来做, 面试和实际工作都会遇到。

场景 1: loss 一开始就是 NaN。 学习率超过 $2/\lambda_{\max}$ (§1.3); 初始化方差错 (§5); fp16 溢出 (Day 1 §10.2); 损失里有 $\log 0$ (Day 1 §5.2.5, 自己写 softmax+log 的典型后果)。

场景 2: loss 前几百步就发散。 没有 warmup (§6.1); Post-LN 架构 (§4.5); 开了动量但没调小学习率 (§2.2 的 $1/(1-\mu)$ 放大)。

场景 3: loss 一直平, 完全不降。 忘了 `zero_grad()` 或 `optimizer.step()`; 参数没注册进优化器 (Day 1 §A.1); 学习率太小; 激活塌缩或死亡 (§5.2 的 PyTorch 默认初始化、dead ReLU)。

场景 4: loss 剧烈震荡但总体下降。 学习率略大或 batch 太小 (§1.6); 没有梯度裁剪 (Day 1 §10.3); 该上动量了 (§2.2)。

场景 5: 训练 loss 很好, 验证 loss 早早上翘。 过拟合。weight decay (要用 AdamW 而不是 Adam+L2, §2.7)、dropout、数据量、早停。

§A 优化器 / 调度器 / 混合精度: API 用法与边界

每一条都对应前面的某个原理。理论错了训练慢; 这些 API 错了训练直接废掉, 而且常常不报错。

A.1 参数分组: 把 bias 和 norm 排除出 weight decay

§2.7 结尾说过 bias 和 LayerNorm 的 γ, β 不该被衰减。实现靠参数分组:

```

decay, no_decay = [], []
for name, p in model.named_parameters():
    if not p.requires_grad:
        continue
    if p.ndim <= 1 or name.endswith(".bias"):      # bias、norm 的  $\gamma/\beta$  都是 1
        no_decay.append(p)
    else:
        decay.append(p)
  
```

```
opt = torch.optim.AdamW([
    {"params": decay, "weight_decay": 0.1},
    {"params": no_decay, "weight_decay": 0.0},
], lr=3e-4, betas=(0.9, 0.95))
```

`p.ndim <= 1` 这个判据很实用：所有 bias 和 norm 的仿射参数都是 1 维，所有权重矩阵都是 2 维以上。nanoGPT 等开源实现用的就是它。

边界：每个参数只能属于一个组，重复加入会报 *"some parameters appear in more than one parameter group"*。

A.2 zero_grad(set_to_none=True)

PyTorch 2.x 起是默认值。带动量的优化器在 `grad is None` 时完全跳过该参数，而"零梯度"时仍会用历史动量更新它，冻结部分参数时两者结果不同。老代码的 `p.grad.zero_()` 在新默认下会 `AttributeError`。详见 Day 1 §A.6。

A.3 调度器：调用位置和频率

```
for step in range(total_steps):
    ...
    opt.step()
    sched.step()          # ★ 必须在 opt.step() 之后
```

1. `sched.step()` 必须在 `opt.step()` 之后，反了的话第一个 step 用的学习率是错的。
2. 按 `step` 调度就每步调用，按 `epoch` 调度就每 `epoch` 调用。`CosineAnnealingLR(T_max=100)` 里的 100 是你调用 `step()` 的次数，它不知道你在什么粒度上调用。搞混的后果是曲线被压缩或拉伸几十倍。
3. 读当前学习率用 `sched.get_last_lr()`（每个 `param_group` 一个值）或直接读 `opt.param_groups[0]['lr']`，不要用 `get_lr()`，那是内部方法，在 `step()` 之外调用会给出错误的值并警告。

A.4 warmup + cosine 的两种写法

写法一：组合现成调度器

```
from torch.optim.lr_scheduler import LinearLR, CosineAnnealingLR,
SequentialLR
warm = LinearLR(opt, start_factor=0.01, total_iters=WARMUP)
cos = CosineAnnealingLR(opt, T_max=TOTAL - WARMUP, eta_min=PEAK_LR * 0.1)
sched = SequentialLR(opt, [warm, cos], milestones=[WARMUP])
```

`milestones` 是切换发生的 `step` 序号，个数比调度器少 1。

写法二：LambdaLR 手写（大模型训练脚本常用，完全可控）

```
import math
def lr_lambda(step):
    if step < WARMUP:
        return step / max(1, WARMUP) # ★ max(1,·) 防
WARMUP=0 除零
    prog = min(1.0, (step - WARMUP) / max(1, TOTAL - WARMUP)) # ★ 夹住, 防
止超出总步数
    return 0.1 + 0.45 * (1 + math.cos(math.pi * prog)) # 从 1.0 衰减到
0.1
sched = torch.optim.lr_scheduler.LambdaLR(opt, lr_lambda)
```

LambdaLR 返回的是"乘在初始 lr 上的系数", 不是学习率本身。这是最常见的误用: 返回 $3e-4$ 会得到 $3 \times 10^{-4} \times 3 \times 10^{-4}$ 。

OneCycleLR 的坑: `total_steps` 必须和实际步数精确匹配, 多跑一步就抛 *"Tried to step N+1 times. The specified number of total steps is N"*。用 DataLoader 时注意 `drop_last` 会改变每 epoch 的步数。

A.5 混合精度: 顺序错一步就白干

```
from torch.amp import autocast, GradScaler
scaler = GradScaler() # fp16 需要; bf16 不需要

for x, y in loader:
    opt.zero_grad(set_to_none=True)
    with autocast(device_type="cuda", dtype=torch.float16):
        loss = criterion(model(x), y) # ★ 只包前向和 loss
        scaler.scale(loss).backward() # 放大 loss 再反传 (Day 1
§10.2)
    scaler.unscale_(opt) # ★★ 裁剪前必须先还原梯度尺度
    nn.utils.clip_grad_norm_(model.parameters(), 1.0)
    scaler.step(opt) # 内部检查 inf/nan, 有则跳过这一步
    scaler.update() # 动态调整放大倍数
```

scaler.unscale_(opt) 是整段最容易漏的一行。不写的话, `clip_grad_norm_` 看到的是被放大几百上千倍的梯度, 范数远超 1.0, 每一步都被狠狠裁剪, 实际学习率被砍到约 $1/\text{scale}$ 。训练看起来在跑, 几乎不动, 完全不报错。

- `autocast` 只包前向和 loss, 不要把 `backward()` 和 `step()` 包进去。
- `unscale_` 对同一个 optimizer 每步只能调一次。
- `scaler.step()` 遇到 inf/nan 会跳过这次更新, 所以日志里偶尔看到 scale 下降是正常的, 持续下降才有问题。
- 用 bf16 时整个 **GradScaler** 都不需要 (Day 1 §10.2: bf16 的指数位和 fp32 一样宽)。直接 `autocast(dtype=torch.bfloat16)`, 省掉上面一半的复杂度。这是 bf16 胜出的又一个实际理由。

A.6 nn.LayerNorm 的 normalized_shape

```
nn.LayerNorm(512)           # 对最后一维 (512) 归一化 — NLP 标准用法
nn.LayerNorm([8, 512])      # 对最后两维一起归一化
```

参数是“要归一化的那几个末尾维度”。传 `[seq_len, d]` 会把序列维卷进统计量，变成一个和序列长度绑定的操作，变长输入直接崩，也丢掉了 §4.3 说的 LayerNorm 全部优点。

`eps` 默认 $1e-5$ （加在方差上，§4.3 的 $\sigma = \sqrt{v + \epsilon}$ ）；`elementwise_affine=False` 去掉 γ, β ；新版 PyTorch 有 `nn.RMSNorm`，老版本要自己写（§4.4 的公式）。

A.7 nn.init: 几个容易记错的默认值

```
nn.init.kaiming_normal_(w, mode="fan_in", nonlinearity="relu")  # He
                                                                    (§5.1)
nn.init.xavier_uniform_(w, gain=nn.init.calculate_gain("tanh"))
nn.init.normal_(w, mean=0.0, std=0.02)                          #
Transformer 常用
```

- `mode="fan_in"` 保持前向二阶矩，`mode="fan_out"` 保持反向（§5.1 末尾的镜像推导）。默认 `fan_in`。
- `kaiming_*` 默认 `nonlinearity="leaky_relu"`、`a=0`，此时增益 $\sqrt{2/(1+0^2)} = \sqrt{2}$ ，和 ReLU 相同，所以默认值对 ReLU 是对的。激活是 `tanh` 或线性时必须显式改，否则增益差 $\sqrt{2}$ 倍。
- `nn.Linear` 的默认初始化是 `kaiming_uniform_(a=√5)`，结果是 $\mathcal{U}(\pm 1/\sqrt{\text{fan_in}})$ ，方差 $1/(3 \text{ fan_in})$ ，只有 He 的 $1/6$ （§5.2 实测 10 层缩到 10^{-7} ）。浅网络够用，深网络必须自己初始化。

A.8 可复现性

```
torch.manual_seed(0); np.random.seed(0); random.seed(0)
torch.backends.cudnn.deterministic = True           # 可复现，但慢
torch.backends.cudnn.benchmark = False              # 与上面互斥
```

- `benchmark=True` 在输入尺寸固定时能明显加速（为该尺寸挑最快的卷积算法），尺寸变化频繁时反而更慢。
- 完全确定性还需要 `torch.use_deterministic_algorithms(True)`，某些算子没有确定性实现会直接报错，并提示设 `CUBLAS_WORKSPACE_CONFIG`。
- `DataLoader` 的 `num_workers>0` 有独立的随机状态，要复现得传 `worker_init_fn` 和 `generator`。调试时先设 `num_workers=0` 排除这个变量。

§B 实操

1. 跑通 `day2_checks.py`，九组结论逐一看懂。特别留意第 3 组（ κ vs $\sqrt{\kappa}$ ）和第 6 组（Pre-LN/Post-LN 随深度的变化），这两组最能说明“为什么”。
2. 从零实现 LayerNorm 的前向和反向（不用 `nn.LayerNorm`），按 §4.3 的三项公式写反向，和 autograd 对拍。参考：`code/d2_layernorm_autograd.py`（`autograd.Function` 手写反向 +

gradcheck, 顺带 RMSNorm 和雅可比投影的数值验证)。

3. 复现五个诊断场景 (§7)：拿 Day 1 附录的 MLP 脚本，逐一制造：lr 调到 10、初始化 std 改成 100、用 PyTorch 默认初始化堆 10 层 ReLU、注释掉 `zero_grad()`、把 AdamW 换成 `Adam(weight_decay=0.1)`。每个病症亲手制造一遍，以后一眼就认得出。参考：
`code/d2_diagnostics.py --scenario all`，七个场景各有"病"和"治"两个版本，判定由实际曲线给出。
4. 加分题：在一个二维病态二次型上画出 GD、Momentum、Adam 的轨迹，看峡谷里的"横跳"和动量的"顺谷而下"。参考：`code/d2_optimizers_numpy.py` (8 个优化器 + 牛顿法)；再跑 `--compare-angles`，看把峡谷转 45° 后 AdaGrad/Adam 的优势为什么消失 (§1.5 的"对角尺子")。

全部参考实现和一键自检见 `code/README.md`。

§C 自测

动机与直觉层

1. "梯度是最陡方向"在什么前提下成立？在 L_∞ 范数和 P 范数下，最陡方向分别是什么？
2. 为什么梯度下降的学习率上界由 $\lambda_{\max}$ 决定，而收敛速度由 κ 决定？画出峡谷的几何图像。
3. 牛顿法为什么是"理想"？它的三个致命问题是什么？用"尺子"视角说明 Adam 在近似什么。
4. 动量为什么能把 κ 降成 $\sqrt{\kappa}$ ？(提示：二阶递推、复根、模长与 λ 无关)
5. Adam 的 $\sqrt{v}$ 是不是 Hessian 对角的近似？为什么说它更像"带平滑的符号下降"？
6. BatchNorm 的三个问题分别是什么？为什么它们在 NLP 里都是致命的？
7. Pre-LN 和 Post-LN 的梯度随深度怎么变化？为什么 Post-LN 更依赖 warmup？Pre-LN 付出了什么代价？

推导层

8. 从二阶泰勒展开推出沿梯度方向的最优步长 $\eta^* = g^\top g / g^\top H g$ 。
9. 推出 $\mathbb{E}[m_t] = \bar{g}(1 - \beta_1^t)$ ，说明偏差修正因子的来历。
10. 证明 Adam 的第一步长度约等于 η ，与梯度大小无关。什么时候这个结论失效？
11. 推导 He 初始化的 $2/n$ 。为什么必须跟踪二阶矩而不是方差？因子 2 从哪来？
12. 索引级推出 LayerNorm 的反向，指出三项分别对应哪条路径。为什么 $\partial v / \partial x_j$ 里的第二项会消失？
13. 说明 LayerNorm 的雅可比为什么是一个投影，它投影掉了哪两个方向，为什么。
14. 写出 Adam+L2 的实际衰减量，说明它为什么依赖梯度大小而 AdamW 不。

工程与 API 层

15. 7B 模型用 AdamW，优化器状态占多少显存？SGD+Momentum 呢？8-bit 优化器呢？
16. `torch.optim.Adam(weight_decay=0.1)` 和 `AdamW(weight_decay=0.1)` 有什么区别？
17. 混合精度训练里 `scaler.unscale_()` 漏写会怎样？为什么不报错？bf16 为什么不需要它？
18. `LambdaLR` 的函数应该返回什么？`CosineAnnealingLR` 的 `T_max` 是什么单位？
19. `nn.Linear` 的默认初始化方差是多少？深网络直接用它会怎样？
20. §7 的五个场景，各说出至少三个排查方向。

出关标准

- 能用柯西-施瓦茨推出"欧氏尺子下梯度最陡"，并说出 L_∞ 、 P 范数下的最陡方向

- ☐ 能从二阶泰勒推出最优步长和牛顿步，说清牛顿法为什么用不了
 - ☐ 能用"尺子"视角把 SGD、Adam、符号下降、牛顿法、Muon 放进同一张表
 - ☐ 能推出动量把 κ 降为 $\sqrt{\kappa}$ 的关键步骤
 - ☐ 能沿 SGD $\rightarrow$ AdamW 讲出每一步解决的具体问题
 - ☐ 能推出 Adam 的偏差修正，并说明第一步长度为什么约等于 η
 - ☐ 能说清 AdamW 和 Adam+L2 的区别
 - ☐ 能索引级推出 LayerNorm 反向，并说出其雅可比是投影、投影掉了什么
 - ☐ 能说出 RMSNorm 少掉了哪一项
 - ☐ 能用实测数据解释 Pre-LN 与 Post-LN 的差异、warmup 的依赖、以及 Pre-LN 的代价
 - ☐ 能推出 He 初始化，并说出 PyTorch 默认初始化的问题
 - ☐ 能说出 warmup 的两个独立理由
 - ☐ 能把现代 LLM 训练配方逐条还原成它要解决的问题
 - ☐ SA 的 API 坑能说出现象和正确写法（尤其 AMP 的 `unscale_` 顺序）
-

疑问清单

-
-
-