# 模型预训练

Jiachen Li

Wuhan University

Nov 13, 2025

目前，常用的预训练任务主要分为三类，包括语言建模（Language Modeling, LM）、去噪自编码（Denoising Autoencoding, DAE）以及混合去噪器（Mixture-of-Denoisers, MoD）

语言建模任务是目前绝大部分大语言模型泛滥应用的预训练任务。该任务的核心在于“预测下一个词元”，并且已经常被应用于训练基于编码器的大语言模型。

给定一个词序列 $u = \{u_1, \ldots, u_T\}$，语言建模任务的目标定义为预测词元：基于序列中当前位置信息之前的词元序列 $u_{<t}$，采用自回归的方式对于目标词元 $u_t$ 进行预测。

在训练过程中，模型通常根据以下的似然函数进行优化：

$$\mathcal{L}_{LM}(u) = \sum_{t=1}^{T} \log P(u_t | u_{<t}).$$

重要变种：前缀语言模型（Prefix Language Modeling）：训练阶段，每个文本序列 $u$ 会根据随机选择的位置 $k$ $(1 \leq k \leq T)$ 切分为前缀 $u_{prefix} = \{u_1, \ldots, u_k\}$ 和后缀 $u_{suffix} = \{u_{k+1}, \ldots, u_T\}$ 两个部分。

在这种情况下，模型的目标被定义为：

$$\mathcal{L}_{Prefix}(u) = \log P(u_{suffix} | u_{prefix}) = \sum_{t=k+1}^{T} \log P(u_t | u_{<t}).$$

语言建模的另一个重要变种是中间填充任务。此任务通过重新调整输入序列的顺序，旨在训练模型对于中间缺失信息的填充能力。具体来说，一个输入序列 $u$ 被划分为三个部分：前缀 $u_{prefix}$、中间部分 $u_{middle}$ 和后缀 $u_{suffix}$。随后，中间部分将被移至序列末尾。因此，模型需要自回归地对新序列 $u_{prefix} \oplus u_{suffix} \oplus u_{middle}$ 进行预测。通过这种方式，模型能够学习填充中间缺失信息的能力。这种任务的训练函数可表示如下：

$$\mathcal{L}_{FIM}(u) = \log P(u_{prefix}) + \log P(u_{suffix}|u_{prefix}) + \log P(u_{middle}|u_{prefix}, u_{suffix}).$$

除了传统的语言建模任务外，去噪自编码任务是另一种常见的语言模型预训练任务，广泛应用于BERT、T5等预训练语言模型中。在去噪自编码任务中，输入文本经过一系列随机替换或删除操作，形成损坏的文本$u_{\tilde{l}}$。模型的目标是根据这些损坏的文本恢复被替换或删除的词元片段$\tilde{u}$。去噪自编码器的训练目标可以用以下数学公式表示：

$$\mathcal{L}_{DAE}(u) = \log P(\tilde{u}|u_{\tilde{u}}).$$

- 通过将语言建模和去噪自编码的目标规划为不同时类型的去噪任务，对预训练任务进行了统一建模。具体来说，混合去噪器定义了三种去噪器：$S$-去噪器、$R$-去噪器和 $X$-去噪器。
- $S$-去噪器与前缀语言模型的目标相同（如公式 6.2 所示），旨在训练模型学习基于给定前缀信息生成合理的后缀文本的能力。

  模型的目标函数可表示为：

$$\mathcal{L}_{LM}(u) = \sum_{t=1}^{T} \log P(u_t | u_{<t}).$$

- $R$-去噪器和 $X$-去噪器与去噪自编码任务的优化目标更为相似：

$$\mathcal{L}_{DAE}(u) = \log P(\tilde{u}|u_{\bar{u}}).$$

- 二者仅在被覆盖片段的跨度和损坏比例上有所区别。$R$-去噪器屏蔽序列 $u_{\sim}$ 中约 15% 的词元，且每个被屏蔽的片段仅包含 3 到 5 个词元。而 $X$-去噪器则采用更长的片段（12 个词元以上）或更高的损坏比例（约 50%），进而要求模型能够精准还原原始信息。这种设置增加了任务难度，迫使模型学习到更全面的文本表示。
- 为了引导模型针对不同类型的输入选择相应的去噪器，输入时会以特殊词元（如 $[R], [S], [X]$）作为开头。

与传统神经网络的优化类似，通常使用批次梯度下降算法来进行模型参数的调优。同时，通过调整学习率以及优化器中的梯度修正策略，可以进一步提升训练的稳定性。为了防止模型对数据产生过度拟合，训练中还需要引入一系列正则化方法

- 在大模型预训练中，通常将批次大小（Batch Size）设置为较大的数值
- 为了更好地训练大语言模型，现在很多工作都采用了动态批次调整策略，即在训练过程中逐渐增加批次大小，最终达到百万级别。例如，GPT-3 的批次大小从 32K 个词元逐渐增加到 3.2M 个词元
- 动态调整批次大小的策略可以有效地稳定大语言模型的训练过程。这是因为较小的批次对应反向传播的频率更高，训练早期可以使用少量的数据让模型的损失尽快下降；而较大的批次可以在后期让模型的损失下降地更加稳定，使模型更好地收敛。

- 现有的大语言模型在预训练阶段通常采用相似的学习率调整策略，包括预热阶段和衰减阶段。预热阶段一般占整个训练步骤的 0.1 开始进行衰减。
- 在模型训练的初始阶段，由于参数是随机初始化的，梯度通常也比较大，因此需要使用较小的学习率使得训练较为稳定

- 训练中通常采用线性预热策略来逐步调整学习率。具体来说，学习率将从一个非常小的数值（例如0或者$1\times10-8$）线性平稳增加，直到达到预设的最大阈值。模型在学习率较大时可以加快收敛速度。达到最大阈值之后学习率会开始逐渐衰减，以避免在较优点附近来回震荡。最后，学习率一般会衰减到其最大阈值的10
- 常见的衰减策略有线性衰减，余弦衰减，平方根倒数衰减，

在已有的工作中，大语言模型的训练通常采用 Adam 及其变种 AdamW 作为优化器。Adam 优化器使用梯度的“动量”作为参数的更新方向，它使用历史更新步骤的梯度加权平均值来替代当前时刻的梯度，从而缓解梯度带来的损失震荡。进一步，Adam 使用自适应的学习率方法，通过梯度的加权二阶矩来对梯度进行修正（可以看做做出“标准差”进行“归一化”），从而防止梯度过小导致模型难以优化。Adam 在优化中引入了三个超参数，在大模型训练中通常使用以下设置：$\beta_1 = 0.9, \beta_2 = 0.95$ 和 $\epsilon = 10^{-8}$。此外，谷歌的研究者提出了 Adafactor 优化器，它是 Adam 优化器的一个变种，通过引入了特异设计可以在训练过程中节省显存，被用于 PaLM 和 T5 等大语言模型的训练。Adafactor 常见的超参数设置如下：
$\beta_1 = 0.9, \beta_2 = 1.0 - k^{-0.8}$，其中 $k$ 表示训练步数。

为方便理解，我们先介绍其他算法。

**Algorithm 1** Adam (Adaptive Moment Estimation)

初始值 $\phi_0$, 学习率 $\alpha$, 衰减率 $\beta_1$ 和 $\beta_2$, 常数 $\epsilon$ 初始化一阶动量 $m = 0$ 和二阶动量 $v = 0$;

**for** $t = 1 : T$ **do** 计算梯度 $g_t = \sum_{i \in B_t} \nabla_{\phi_i} \mathcal{L}(\phi_t)$;
一阶动量 $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$;
二阶动量 $v_t = \beta_2 v_{t-1} + (1 - \gamma) g_t \odot g_t$;
调整动量 $\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$;
$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$;
参数更新 $\phi_{t+1} = \phi_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \odot \hat{m}_t$;

**Algorithm 2** 带动量的随机梯度下降（SGD with Momentum）

初始值 $\phi_0$, 学习率 $\alpha$, 衰减率 $\rho$
初始化一阶动量 $m = 0$;

**for** $t = 1 : T$ **do** 计算梯度：$g_t = \sum_{i \in B_t} \nabla_{\phi_i} \mathcal{L}(\phi_t)$;
计算一阶动量：$m_t = \rho m_{t-1} + (1 - \rho) g_t$;
更新参数：$\phi_{t+1} = \phi_t - \alpha m_t$;

**Algorithm 3** 梯度归一化 (Normalized Gradient)

初始值 $\phi_0$, 学习率 $\alpha$, 常数 $\epsilon$

**for** $t = 1 : T$ **do** 计算梯度：$g_t = \sum_{i \in B_t} \nabla_{\phi_i} \mathcal{L}(\phi_t)$;
计算二阶动量：$v_t = g_t \odot g_t$;
更新参数：$\phi_{t+1} = \phi_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} \odot g_t$;

**Algorithm 4** Adam (Adaptive Moment Estimation)

初始值 $\phi_0$, 学习率 $\alpha$, 衰减率 $\beta_1$ 和 $\beta_2$, 常数 $\epsilon$
初始化一阶动量 $m = 0$ 和二阶动量 $v = 0$;

**for** $t = 1 : T$ **do** 计算梯度 $g_t = \sum_{i \in B_t} \nabla_{\phi_i} \mathcal{L}(\phi_t)$;
一阶动量 $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$;
二阶动量 $v_t = \beta_2 v_{t-1} + (1 - \gamma) g_t \odot g_t$;
调整动量 $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$;
$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$;
参数更新 $\phi_{t+1} = \phi_t - \frac{\alpha}{\sqrt{\hat{v}_t + \epsilon}} \odot \hat{m}_t$;

**Algorithm 5** Adafactor for weight vectors

初始点 $X_0 \in \mathbb{R}^n$，相对步长 $\{\rho_t\}_{t=1}^T$，二阶动量衰减 $\{\hat{\beta}_2\}_{t=1}^T$ 使得 $\hat{\beta}_{21} = 0$，正则化常数 $\epsilon_1$ 和 $\epsilon_2$，截断阈值 $d$

**for** $t = 1\ T$ **do** $\alpha_t = \max(\epsilon_2, \mathrm{RMS}(X_{t-1}))\rho_t$

$G_t = \nabla f_t(X_{t-1})$

$\hat{V}_t = \hat{\beta}_2 \hat{V}_{t-1} + (1 - \hat{\beta}_2)(G_t^2 + \epsilon_1 \mathbf{1}_n)$

$U_t = \frac{G_t}{\sqrt{\hat{V}_t}}$

$\hat{U}_t = \frac{U_t}{\max(1, \frac{1}{d})}$

$X_t = X_{t-1} - \alpha_t \hat{U}_t$

在大语言模型的训练过程中，经常会遇到训练不稳定的问题。下面介绍几种深度学习中常用的稳定训练技术

- 训练中一种常见的现象是损失的突增。为了解决这一问题，可以采取梯度裁剪（Gradient Clipping）的方法，把梯度限制在一个较小的区间内：当梯度的模长超过给定的阈值后，便按照这个阈值进行截断。在大模型训练中，这个阈值通常设置为 1.0。
- 如果 ||g||>c，则更新梯度 g 为 $g \times c / ||g||$ 其中 ||g|| 是梯度向量的范数，c 是预设的阈值

- 在模型的训练过程中也通常会引入正则化技术来稳定训练过程，提高模型的泛化能力。AdamW 中采用了权重衰减（Weight Decay）方法，在每次更新模型参数时引入衰减系数，这个系数通常设置为 0.1。
- 模型原有的损失函数我们记为 $l(w,b)$，我们增加了 L2 惩罚项，新损失函数记为：

$$l(w,b)+\lambda\|w\|^2$$

其中 $\lambda>0$，当 $\lambda=0$ 时惩罚项不起作用，可以预防训练时 $w$ 过大造成过拟合。

为了进一步避免训练过程的异常情况，另一种常用的实践策略是每隔固定的步数设置一些模型存档点。当模型发生了训练异常时（例如损失激增），便可以选择前一个存档点重启训练过程，并跳过可能导致问题的数据。

主要面临着两个技术问题：一是如何提高训练效率；二是如何将庞大的模型有效地加载到不同的处理器中。在本节中，我们将介绍几种常见的高效训练技术，包括 3D 并行训练、激活重计算和混合精度训练。

- 3D 并行策略实际上是三种常用的并行训练技术的组合，即数据并行（Data Parallelism）、流水线并行（Pipeline Parallelism）和张量并行（Tensor Parallelism）
- 数据并行是一种提高训练吞吐量的方法，它将模型参数和优化器状态复制到多个 GPU 上，然后将训练数据平均分配到这些 GPU 上。这样，每个 GPU 只需要处理分配给它的数据，然后执行前向传播和反向传播以获取梯度。当所有 GPU 都执行完毕后，该策略会将不同 GPU 的梯度进行平均，以得到整体的梯度来统一更新所有 GPU 上的模型参数。
- 朴素的流水线调度并不能达到真正的并行效果。以图 6.3 (d) 为例，1 号 GPU 在前向传播后需要等待 2 号 GPU 反向传播的结果才能进行梯度传播，因此整个流程是“1 号前向-2 号前向-2 号反向-1 号反向”的串行操作。为了解决这一问题，流水线并行通常需要配合梯度累积（Gradient Accumulation）技术进行优化。

该技术的主要思想是，计算一个批次的梯度后不立刻更新模型参数，而是累积几个批次后再更新，这样便可以在不增加显存消耗的情况下模拟更大的批次。在流水线并行中使用了梯度累积后，1号卡前向传播完第一个批次后，便可以不用等待，继续传播第二个和后续的批次，从而提高了流水线的效率

- 张量并行相对于流水线并行的分配粒度更细，它进一步分解了模型的参数张量（即参数矩阵），以便更高效地利用多个 GPU 的并行计算能力。
- 具体地，对于大语言模型中的某个矩阵乘法操作 $WH$，参数矩阵 $W$ 可以按列分成两个子矩阵 $W_1$ 和 $W_2$，进而原式可以表示为 $[W_1, W_2]H$。然后，可以将参数矩阵 $W_1$ 和 $W_2$ 放置在两张不同的 GPU 上。
- 上，然后后续进行两列矩阵乘法操作，最后通过跨 GPU 通信将两个 GPU 的输出组合成最终结果。

- 零冗余优化器（Zero Redundancy Optimizer, ZeRO）技术由 DeepSpeed 代码库提出，主要用于解决在分布式并行中的模型冗余问题，即每张 GPU 需要复制一份模型参数。
- 为了解决这个问题，ZeRO 技术不仅在每个 GPU 上保留部分模型参数和优化器参数，当需要时再从其他 GPU 中获取。如图 6.3 (b) 所示，模型被均分在两张 GPU 上，当需要使用第一层计算时，两张卡分别从对应的模型参数进行计算，使用完之后便可以释放相应显存，从而降低显存占用。

- 是一种用于优化反向传播时显存占用的技术。具体来说，给定一个待优化函数 $Y = XW$，在反向传播时需要 $X$ 的值才能计算 $W$ 的导数，所以在前向传播时需要保留这些 $X$（通常被称为激活）。然而，保留每一层所有的激活值需要占用大量的显存资源（具体的显存占用见 6.4.4 节）。因此，激活重计算技术在前向传播时仅保留部分激活值，然后在反向传播时重新计算这些激活值，以达到节约显存的目的，但是同时也会引入额外的计算开销。
- 在大语言模型的训练过程中，激活重计算的常见方法是将 Transformer 的每一层的输入保存在下来，然后在反向传播时计算对应的激活值。

混合精度训练（Mixed Precision Training）技术，通过同时使用半精度浮点数（2 个字节）和单精度浮点数（4 个字节）进行运算，以实现显存开销减半、训练效率翻倍的效果。

在本节中，我们将介绍如何计算基于 Transformer 架构的大语言模型的参数数量，并给出训练模型时所需要的运算量、训练时间和显存开销估计

- 由于当前主流的大模型普遍采用因果解码器架构，因此下面以 LLaMA 模型为范例，深入分析其参数量计算方式。对于其他模型，其参数量计算算法可参照此方法计算。
- 首先，假设词表大小为 $V$，模型包含 $L$ 层解码器，中间状态的维度大小为 $H$，前馈网络层的中间状态维度大小为 $H'$。
- 我们主要关注计算以下几个部分的参数量：
  - 输入嵌入层：输入嵌入层 $E \in \mathbb{R}^{V \times H}$ 将词表中的每个单词（一共 $V$ 个单词）映射到一个 $H$ 维的向量，因此输入嵌入层有 $V \times H$ 个参数。

- 传统的注意力机制部分包含查询 $W_Q \in \mathbb{R}^{H\times H}$、键 $W_K \in \mathbb{R}^{H\times H}$ 和值 $W_V \in \mathbb{R}^{H\times H}$ 的线性变换矩阵，每个变换矩阵都包含 $H^2$ 个参数，所以这部分需要 $3\times H^2$ 个参数。同时还需要一个额外的线性变换来将多头注意力机制 $h$ 的输出拼接后映射成最终输出 ($W_O \in \mathbb{R}^{H\times H}$)，这又需要 $H^2$ 个参数。因此，多头注意力层总共需要 $4\times H^2$ 个参数。
- 其中，多头注意力中我们设置查询 ($W_Q \in \mathbb{R}^{H/h\times H}$)、键 ($W_K \in \mathbb{R}^{H/h\times H}$) 和值 ($W_V \in \mathbb{R}^{H/h\times H}$)。

## 多头自注意力（续）

- 通常，如果输入 $x_m$ 的维度为 $D$，并且有 $H$ 个头，则值、查询和键的大小都将为 $D/H$，因为这样可以有效节省实现。
- 这些注意力机制的输出进行垂直级联，随后应用另一个线性变换 $\Omega_c$ 来组合它们，即

$$MhSa(X) = \Omega_c \left( (S_1 A_1(x)^T, S_2 A_2(x)^T, \cdots, S_H A_H(x)^T) \right)^T .$$

- LLaMA 的前馈网络层由三个线性变换组成，其中有一个非线性激活函数。前两个线性变换 ($W^U \in \mathbb{R}^{H \times H'}$ 和 $W^G \in \mathbb{R}^{H' \times H'}$) 将输入从 $H$ 维映射到 $H'$ 维，需要 $2 \times H \times H'$ 个参数；最后一个线性变换 ($W^D \in \mathbb{R}^{H' \times H}$) 将输出从 $H'$ 维映射回 $H$ 维，需要 $H' \times H$ 个参数。因此，前馈网络层总共需要 $3 \times H \times H'$ 个参数。

  具体来说，给定输入 $X$，Transformer 中的前馈神经网络由两个线性变换和一个非线性激活函数组成：

$$FFN(X) = \sigma(XW^U + b_1)W^D + b_2, \tag{5.8}$$

- 最后，LLaMA 的输出层包含一个线性变换 $(W^L \in \mathbb{R}^{H \times V})$，将解码器的输出映射到词表大小 $v$ 的维度上，使用 softmax 函数预测下一个单词的概率分布。这个线性变换需要 $V \times H$ 个参数。

******* 综上所述，累积输入嵌入层、输出层和 $L$ 层解码器每层的多头注意力层，前馈网络层和归一化层，**LLaMA** 模型的参数量计算公式为：

$$\text{参数量} = 2VH + H + L \times (4H^2 + 3HH' + 2H).$$

- 浮点运算包括浮点数的加减乘除运算，以及浮点数的指数函数、对数函数、三角函数等运算操作。
- 在分析多头注意力和线性变换的运算量时，我们进一步设定以下参数：模型参数量为 $P$，批处理大小为 $B$，输入序列长度为 $T$，因此训练词元总数量为 $C = BT$；多头注意力机制包含 $N$ 个头，每个头的维度为 $D$，因此中间状态维度 $H$ 满足关系 $H = ND$。其他定义与上一节 6.4.1 保持一致。

小贴士（矩阵乘法运算量）
矩阵 $A \in \mathbb{R}^{n \times m}$ 和矩阵 $B \in \mathbb{R}^{m \times p}$ 附带所需的运算量为 $2nmp$。

- 多头注意力：首先分析多头注意力机制一次计算的运算量。对于批次化的数据，计算得到相应的查询、键和值张量（公式 5.2），$Q, K, V \in \mathbb{R}^{B\times T\times H}$，考虑到需要进行多头计算，这些张量需要进行拆分和转置，得到 $Q', K', V' \in \mathbb{R}^{B\times N\times T\times D}$。在注意力计算中（公式 5.7），$Q'K'T'$ 的矩阵乘法需要 $2BT^2ND$ 次浮点运算；接着，进行标准化操作（$\sqrt{D}$ 归缩）需要 $BT^2N$ 次浮点运算，softmax 操作需要进行指标、加和、归一化操作，总计 $3BT^2N$ 次浮点运算；最后结果与 $V'$ 进行矩阵乘法，再需要 $2BT^2ND$ 次浮点运算。
- 因此，一次多头注意力计算的浮点运算量为 $4BT^2ND + 4BT^2N$。考虑到后向传播的运算量大致为前向传播的两倍，整个模型中多头注意力运算量可表示为：

$$\text{运算量} = 12 \cdot (BT^2ND + BT^2N); \quad L = 12CTL \cdot (H + N). \tag{6.6}$$

其中 $T$ 是序列长度，$H$ 是中间状态维度。

- 线性变换：接下来考虑线性变换的训练运算量，其中包括注意力层中的四个映射（公式 5.2, 5.3, 5.4 和 5.6）、前馈网络层的变换（公式 5.8）以及输出层映射（公式 5.11）。以前馈网络层中的上映射操作 $X'W^U$ 为例，中间状态 $X' \in \mathbb{R}^{B\times T\times H}$，因此其前向传播需要 $2BT'HH'$ 次浮点运算，反向传播则需要 $4BTHH'$ 次浮点运算，总计需要 $6CHH'$ 次浮点运算，其中 $C = BT$ 为训练词元总数。可以看到，线性变换的运算量与矩阵参数量相关，因此 Transformer 中所有线性变换部分的运算量公式可表达为：

$$\text{运算量} = 6C : \text{线性变换的参数量}. \tag{6.7}$$

若训练过程中采用了激活重计算技术（第 6.3 节），反向传播时需要额外进行一次前向传播，则总运算量将变为：

$$\text{运算量} = 8C : \text{线性变换的参数量}. \tag{6.8}$$

最后，通过对比公式 6.5、6.6 和 6.7，可以发现多头注意力的运算量约为线性变换运算量的 $\frac{T}{6H}$，考虑到大模型训练时序列长度 $T$ 小于等于中间状态维度 $H$，因此多头注意力运算量最大为线性变换运算量的 6，其影响相对较小。根据公式 6.5，线性变换的参数量通常占总参数量的 95% 以上。因此，可以直接用参数量 $P$ 替换公式 6.7 中的线性变换的参数量。在这种情况下，参数量 $P$ 的模型在训练词元上进行预训练的总运算量可以按如下公式进行估计：

$$\text{运算量} \approx 6CP. \tag{6.9}$$

进一步，如果使用了激活重计算技术，则运算总量将变为 $8CP$。

$$\text{参数量} = 2VH + H + L \cdot (4H^2 + 3HH' + 2H). \tag{6.6}$$

$$\text{多头注意力运算量} = 12 \cdot (BT^2ND + BT^2N). \tag{6.5}$$

$$\text{线性变换运算量} = C : \text{线性变换的参数量}. \tag{6.7}$$

具体的估计公式如下：

$$\text{训练时间} = \frac{\text{运算量}}{\text{GPU 数量} \times \text{GPU 每秒浮点运算数}}.$$

- 以 LLaMA (65B) 的预训练为例，其参数量 $P = 6.5 \times 10^{10}$，词元数 $C = 1.4 \times 10^{12}$，由于采用了激活重计算技术，其运算量大致为 $8CP = 7.28 \times 10^{23}$。它在预训练过程中使用了 2,048 张 A100 GPU，其每张 A100 GPU 每秒最多能进行 $3.12 \times 10^{14}$ 次 BF16 浮点运算。我们假设在训练过程中，每张 GPU 能达到每秒 $2 \times 10^{14}$ 次 BF16 浮点运算的实际性能。根据上文公式，可以计算出 LLaMA (65B) 使用 2,048 张 A100 GPU 在 1.4T 词元上的训练时间大致为 $1.78 \times 10^6$ 秒，即大约为 20.6 天。这个估算结果与论文中公布的 21 天基本一致。

# Thanks!