lora

来源：大猿搬砖简记 (写得特别细特别好！)

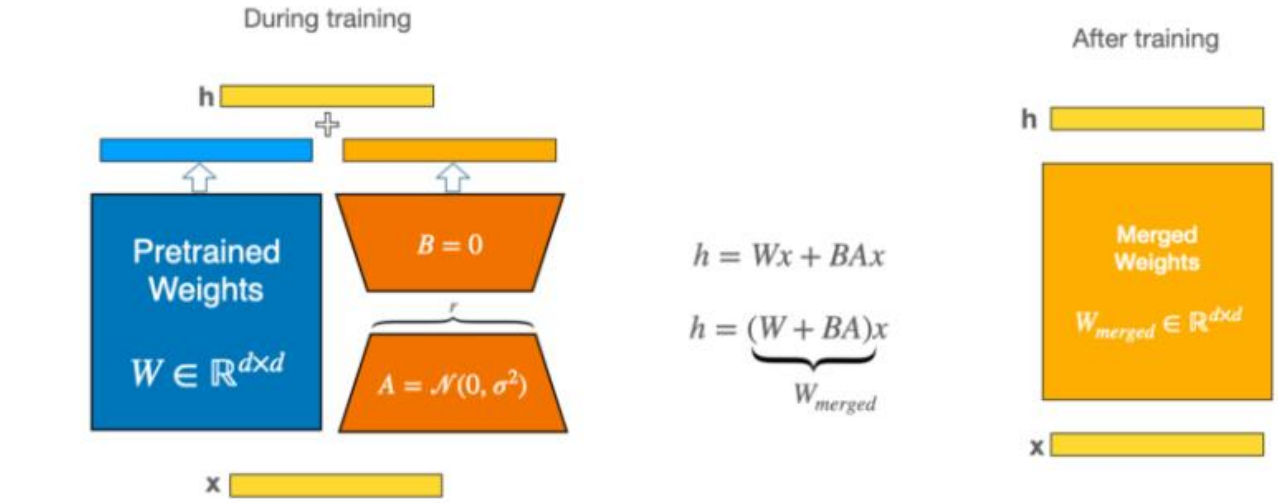
LoRA的目标是 通过低秩矩阵来减少微调过程中的参数量，从而降低计算资源的需求。

lora中低秩矩阵的初始化：

A做一个标准正态分布的随机初始化，B初始化为0。为什么要这么做？

第一批数据喂进来的时候，让我的权重矩阵和不加BA时一样。

motivation：应该尽量减少刚开始训练时的波动，否则会偏离原始矩阵很多，从而导致结果就和原来的偏离很多，当模型层数深了之后就可能有一个传播的效应。会带来训练不稳定。



$$h = W_0x + \Delta Wx = W_0x + BAx$$

https://arxiv.org/pdf/2106.09685

相对于adapter这种串行结构，lora不需要在推理过程中付出任何额外的计算量

在原始论文中lora加在了Wq、Wk、Wv、Wo上，理论上任何涉及矩阵乘法的地方都可以用lora。

在原论文中提到过对于两个低秩矩阵，会用超参\alpha（一个常数）来做调整，这个超参是作为scaling rate直接和低秩矩阵相乘的，也就是最终的输出为：

W_{\text{new}} = W + \frac{\alpha}{r} (BA)

- W 是原始的预训练权重矩阵。
- A 是一个 r \times d 的矩阵，其中 r 是秩， d 是权重矩阵的维度。
- B 是一个 d \times r 的矩阵。
- BA 是一个 d \times d 的矩阵，它是低秩矩阵 A 和 B 的乘积，用于近似全参数微调中的增量权重。
- \alpha / r 是缩放因子，用于调整低秩矩阵对原始权重的影响程度。

两个可以调的超参数：

1. **超参数 \alpha**：

- \alpha 是一个缩放因子，用于控制低秩矩阵对预训练权重更新的贡献程度。
- 它直接影响到微调过程中新知识（通过低秩矩阵 A 和 B 表示）对模型权重的影响大小。
- 较大的 \alpha 值会增加低秩矩阵在更新中的权重，使得模型更倾向于适应新任务的特征，但也可能引入过拟合的风险。
- 较小的 \alpha 值会使更新更加保守，减少对预训练权重的改动，有助于保持模型在新任务上的泛化能力。

2. **秩 r**：
   - r 定义了低秩矩阵 A 和 B 的秩，即这些矩阵可以表示的线性独立向量的数量。
   - 秩 r 决定了低秩矩阵能够捕捉到的特征维度的数量（可以表示矩阵的信息量）。较小的 r 意味着模型只能学习到更精炼的特征子集，而较大的 r 允许模型学习更多的特征维度，但可能会引入更多的噪声。
   - 在LoRA中，通过设置较小的秩，可以减少模型需要训练的参数数量，从而降低训练成本和提高训练效率。

原论文对 \alpha 的解释：

> We illustrate our reparametrization in Figure 1. We use a random Gaussian initialization for A and zero for B, so $\Delta W = BA$ is zero at the beginning of training. We then scale ΔWx by $\frac{\alpha}{r}$, where α is a constant in r. When optimizing with Adam, tuning α is roughly the same as tuning the learning rate if we scale the initialization appropriately. As a result, we simply set α to the first r we try and do not tune it. This scaling helps to reduce the need to retune hyperparameters when we vary r (Yang & Hu, 2021).

这段话大致意思是说，在我们采用Adam做优化器时，调整 \alpha 的作用就相当于调整learning rate。一般而言，我们把 \alpha 设置为我们第一次做实验时设置的r，然后就把 \alpha 固定下来，之后只调整r即可，这样做的好处是当我们尝试不同的r时，我们不需要再去调整别的超参了。

$$h = Wx + \frac{\alpha}{r}BAx$$

其中，W表示预训练权重（旧知识），$\frac{\alpha}{r}BA$表示增量权重ΔW的近似（新知识）。**理论上说，当r较小时，我们提取的是ΔW中信息含量最丰富的维度，此时信息精炼，但不全面；当r较大时，我们的低秩近似越逼近ΔW，此时信息更加全面，但带来的噪声也越多（含有很多冗余无效的信息）。**

基于这个猜想，当我们第一次做实验时，我们会尽量把r调得大些，例如32、64，并假设在这个秩下，低秩权重已经非常近似ΔW了，因此这时我们设置$\alpha = r$，意味着我们假定LoRA低秩微调的效果和全参数微调持平。

那么接下来，我们肯定就要往小的r进行尝试了。这时我们把α固定住，意味着随着r的减小，$\frac{\alpha}{r}$会越来越大，我们这样做的原因是：

- **当r越小时，低秩矩阵表示的信息精炼，但不全面。我们通过调大$\frac{\alpha}{r}$，来放大forward过程中新知识对模型的影响。**
- **当r越小时，低秩矩阵表示的信息精炼，噪声/冗余信息少，此时梯度下降的方向也更加确信，所以我们可以通过调大$\frac{\alpha}{r}$，适当增加梯度下降的步伐，也就相当于调整learning rate了。**

训练过程

在训练过程中进行的操作是固定住预训练权重，只对低秩矩阵进行训练。在保存权重时，只需保存低秩矩阵部分。这种方法在微调GPT-3 175B时，可以显著降低显存消耗：从1.2TB降至350GB；当 (r = 4) 时，最终保存的模型大小从350GB降低至35MB，大大减少了训练的开销。

总体来看，LoRA（低秩矩阵分解）在微调期间能够显著节省显存。然而，问题在于：**在训练的每一时刻，LoRA是否都能节省显存?**

W_{\text{new}} = W + \frac{\alpha}{r} (BA)

在反向传播时，我们需要计算损失函数 L 对 W_{\text{new}} 的梯度。根据链式法则，梯度计算可以表示为：

\frac{\partial L}{\partial W} = \frac{\partial L}{\partial W_{\text{new}}} \cdot \frac{\partial W_{\text{new}}}{\partial W}

注意\frac{\partial L}{\partial W_{\text{new}}} 这一项，你会发现，它和预训练权重的维度`d*d`一模一样，也就是为了计算的梯度，我们需要用到和全参数微调过程中一样大小的中间值结果。**因此对LoRA来说，这一层的峰值显存，和全量微调基本是一致的**）。

但是为什么LoRA又能从整体上降低显存使用呢，因为：

- LoRA并不是作用在模型的每一层，例如论文里的LoRA只作用在attention部分。
- LoRA虽然会导致某一层的峰值显存高于全量微调，但计算完梯度后，这个中间结果就可以被清掉了，不会一致保存。
- 当待训练权重从`d*d`降为`2*r*d`时，需要保存的optimizer states也减少了（那可是fp32）。

推理过程

推理过程不会像Adapter那样产生推理上的延时。

在切换不同下游任务时，可以灵活从中移除低秩权重的部分。

每次微调结束后，也不一定要把低秩权重合进W中，可以将“预训练权重”和“低秩权重”分开存储。