AdaLoRA详解

一、背景

论文标题：ADAPTIVE BUDGET ALLOCATION FOR PARAMETER- EFFICIENT FINE-TUNING

论文链接：https://arxiv.org/pdf/2303.10512.pdf

参考链接: https://zhuanlan.zhihu.com/p/657130029

1. LORA缺陷

LORA是一种有效的低资源方式，仅仅微调万分之一参数就能达到全量参数微调的效果。但本文指出**LORA是将可微调参数平均分布在每个权重矩阵上**，**即预先规定了每个增量矩阵Δ的秩r必须相同。忽略了不同层、不同类型参数对下游任务的重要程度，因此LORA微调的效果可能不是最优的。**

2. AdaLORA的提出

AdaLoRA 改进了LORA可微调参数的分配方式，根据每个参数的重要程度自动得为其分配可微调参数的预算。LoRA中是让模型学习BA，去近似SVD分解的结果，但是在训练过程中，没有引入任何**SVD分解相关的性质**做约束，而AdaLoRA则是直接将这一束缚考虑到了Loss中。

具体地：

- 基于奇异值分解(SVD)的形式参数化增量更新，将增量矩阵以奇异值分解的形式表达，规避了大量SVD运算；
- 基于设计的**重要程度的参数分配**（importance-aware rank allocation）方法，来高效裁剪不重要奇异值，从而减少计算开销

核心Idea：AdaLoRA adjusts the **rank** of incremental matrices to control their budget.

二、AdaLORA原理

1. 研究方法

AdaLoRA将增量矩阵的Δ替换为$P\Lambda Q$,这样既省去到了复杂的SVD计算又能显式的裁剪奇异值。

$$W = W^{(0)} + \Delta = W^{(0)} + P\Lambda Q$$

其中Λ以全0初始化，P和Q通过高斯分布初始化，确保训练开始时Δ为0。同时，为保证P和Q的正交性，还在训练过程中增加了一个正则化，保证:

$$P^T P = Q^T Q = I$$

正则化体现在损失函数中:

$$\mathcal{L}(\mathcal{P}, \mathcal{E}, \mathcal{Q}) = \mathcal{C}(\mathcal{P}, \mathcal{E}, \mathcal{Q}) + \gamma \sum_{k=1}^{n} R(P_k, Q_k)$$
$$R(P_k, Q_k) = ||P^T P - I||_F^2 + ||Q^T Q - I||_F^2$$

其中γ>0 ,是正则化的系数，C是训练损失函数，R是正则化函数。

$\mathcal{C}(\mathcal{P}, \mathcal{E}, \mathcal{Q})$ **表示预测结果和真实标签间的差异，P和Q都必须是正交矩阵，即**$P^T P = I, Q^T Q = I$

2. 方法涉及三个具体问题:

- AdaLoRA 中**Loss**的设计（上面讲了）
- AdaLoRA 中**重要性分数计算**的设计
- AdaLoRA 中**如何根据重要性分数筛选不重要的三元组，并调节矩阵的秩**

2.1. 如何建模特征的重要性?

AdaLoRA的整体目标是要做**参数预算（budget）**，即忽略不重要的参数，把训练资源给重要的参数，在AdaLoRA中，通过“**变秩**”来实现这种预算的动态分配的。

tips：为什么不直接修改Lora的r值–**因为r代表的是超参数无法动态调整**。

（一）单参数重要性分数设计：

AdaLoRA作者就提出了这样一种计算t时刻，单个模型参数（$s^{(t)}(w_{ij})$）重要性的方法。

首先给出公式，即用敏感性 $\bar{I}^{(t)}(w_{ij})$ 和不确定性 $\bar{U}^{(t)}(w_{ij})$ 的乘积来表示这个特征的重要性：

$$s^{(t)}(w_{ij}) = \bar{I}^{(t)}(w_{ij}) \cdot \bar{U}^{(t)}(w_{ij})$$

1、敏感性

a. **一个最直观的想法就是：去看看这个参数对Loss的影响。**在模型剪枝中，单个参数的**敏感性**被定义为梯度和权重乘积的绝对值，如下式：

$I(w_{ij}) = |w_{ij} \cdot \nabla_{w_{ij}} \mathcal{L}|$ ，其中 w_{ij} 是任意可训练的权重，

$\nabla_{w_{ij}} \mathcal{L}$ 是这个权重的梯度。

b. 在SGD中，这个重要性只是mini-batch的样本反应的重要性，不同step间重要性分数可能会受到mini-batch客观波动的影响，我们可以使用滑动平均思想（代表性：**momentum**）来减轻mini-batch带来的重要性的评估误差，表示为式：

$$\bar{I}^{(t)}(w_{ij}) = \beta_1 \bar{I}^{(t-1)}(w_{ij}) + (1-\beta_1) I^{(t)}(w_{ij})$$

其中 t 代表的是训练步数， $0 < \beta_1 < 1$ 是滑动平均中控制历史记录和当前批次占比的超参数。

2、不确定性

a. 有了重要性，我们可以计算敏感性的**不确定性**（Uncertainty）[4]，不确定性是AdaLoRA的作者在他的另外一个论文Platon[4]中提出的指标，它表示的是敏感性的局部时间变化，定义为 ：

$$U^{(t)}(w_{ij}) = \left| I^{(t)}(w_{ij}) - \bar{I}^{(t)}(w_{ij}) \right|$$

- $I^{(t)}(w_{ij})$是t时刻下，按上面单参数重要性计算公式的单参数重要性
- $\bar{I}^{(t)}(w_{ij})$是前t-1个时刻该单参数重要性的平滑值
- $|I^{(t)}(w_{ij}) - \bar{I}^{(t)}(w_{ij})|$是当前值与平滑值之间的差异。这一项的意义是，你也不能一股脑地去平滑，也要考虑到重要性分数的真实波动情况

对于不确定性，我们最好也对它进行滑动平局：

$$\bar{U}^{(t)}(w_{ij}) = \beta_2 \bar{U}^{(t-1)}(w_{ij}) + (1-\beta_2) U^{(t)}(w_{ij})$$

3、单参数重要性

a. 最终，我们可以使用敏感性 $\bar{I}^{(t)}(w_{ij})$ 和不确定性 $\bar{U}^{(t)}(w_{ij})$ 的乘积来表示这个特征的重要性：

$$s^{(t)}(w_{ij}) = \bar{I}^{(t)}(w_{ij}) \cdot \bar{U}^{(t)}(w_{ij})$$

（二）三元组重要性分数：

- 对于三元组 $\mathcal{G}_{k,i} = \{\boldsymbol{P}_{k,*i}, \lambda_{k,i}, \boldsymbol{Q}_{k,i*}\}$ ，它的重要性是三元组三个值的加权和，权值取决于 d_1 和 d_2

$$\boldsymbol{S}_{k,i} = s(\lambda_{k,i}) + \frac{1}{d_1}\sum_{j=1}^{d_1} s(\boldsymbol{P}_{k,ji}) + \frac{1}{d_2}\sum_{j=1}^{d_2} s(\boldsymbol{Q}_{k,ij})$$

P_{*i}, Q_{i*} 分别表示“第i列”和“第i行”。

即：三元组的重要性分数 = λ 的重要性分数 + P矩阵列中所有元素重要性分数的均值 + Q矩阵行中所有元素重要性分数的均值。取均值的原因，是不希望参数量影响到重要性分数。

2.2. 如何根据重要性自动计算秩 r 的值，进行动态调整矩阵秩?

为了根据重要性计算秩的值，一个最直观的方式是将 r 看做模型的一个参数，然后根据模型的损失值来调整 r **核心是：根据三元组重要性分数，对 Λ 矩阵中相应的 λ 做置0处理。**

给出符号定义:

- $\nabla P_k \mathcal{L}\left(\mathcal{P}^{(t)}, \mathcal{E}^{(t)}, \mathcal{Q}^{(t)}\right)$: 表示第 k 个模块的 P 矩阵在t时刻的梯度
- $\nabla Q_k \mathcal{L}\left(\mathcal{P}^{(t)}, \mathcal{E}^{(t)}, \mathcal{Q}^{(t)}\right)$: 表示第 k 个模块的 Q 矩阵在t时刻的梯度
- $\nabla \Lambda_k \mathcal{L}\left(\mathcal{P}^{(t)}, \mathcal{E}^{(t)}, \mathcal{Q}^{(t)}\right)$: 表示第 k 个模块的 Λ 矩阵在t时刻的梯度

步骤如下：

（1）首先，我们拿 $\nabla_{\Lambda_k}\mathcal{L}(\mathcal{P}^{(t)}, \mathcal{E}^{(t)}, \mathcal{Q}^{(t)})$ ，先更新一波 Λ_k ，即我们有：

$$\widetilde{\Lambda}_k^t = \Lambda_k^t - \eta \nabla_{\Lambda_k} \mathcal{L}(\mathcal{P}^{(t)}, \mathcal{E}^{(t)}, \mathcal{Q}^{(t)})$$

其中， η 是我们的学习率（learning_rate）

注意，这里 $\widetilde{\Lambda}_k^t$ 头上还顶着t时刻的标志，而不是t+1，也就是说，我**们对 Λ 做完梯度更新后的结果，并不是t+1时刻的结果。我们做完置0后，才是t+1时刻的结果。**

（2）接着，我们按以下方式判断 Λ 矩阵中哪些元素应该置0，哪些元素应该保持为梯度更新后的结果：

$$\Lambda_k^{(t+1)} = \mathcal{T}(\tilde{\Lambda}_k^{(t)}, S_k^{(t)}), \text{ with } \mathcal{T}(\tilde{\Lambda}_k^{(t)}, S_k^{(t)})_{ii} = \begin{cases} \tilde{\Lambda}_{k,ii}^{(t)} & S_{k,i}^{(t)} \text{ is in the top-}b^{(t)} \text{ of } S^{(t)}, \\ 0 & \text{otherwise,} \end{cases}$$

2.3. top-b策略

策略总结：

过程就和warm-up非常相似，在训练刚开始，我们逐渐增加top_b，也就是逐渐加秩，让模型尽可能多探索。到后期再慢慢把top_b降下来，直到最后以稳定的top_b进行训练，达到AdaLoRA的总目的：把训练资源留给最重要的参数。

具体细节：

具体来讲，在最开始的 t_i 步，我们给预算一个稍微大的值，让模型快速达到比较好的效果。接下来的 $T - t_f - t_i$ 步，我们通过让秩的预算以三次方的速度逐渐减小，来达到对秩进行剪枝的目的。在最后剩下的 $T - t_f$ 步中，我们稳定秩的大小来让模型效果达到当前秩上的一个局部最优解。

预算的完整计算方式如式：

$$b^{(t)} = \begin{cases} b^{(0)} & 0 \le t < t_i \\ b^{(T)} + \left(b^{(0)} - b^{(T)}\right)\left(1 - \frac{t - t_i - t_f}{T - t_i - t_f}\right)^3 & t_i \le t < T - t_f \\ b^{(T)} & \text{o.w.} \end{cases} \qquad (12)$$

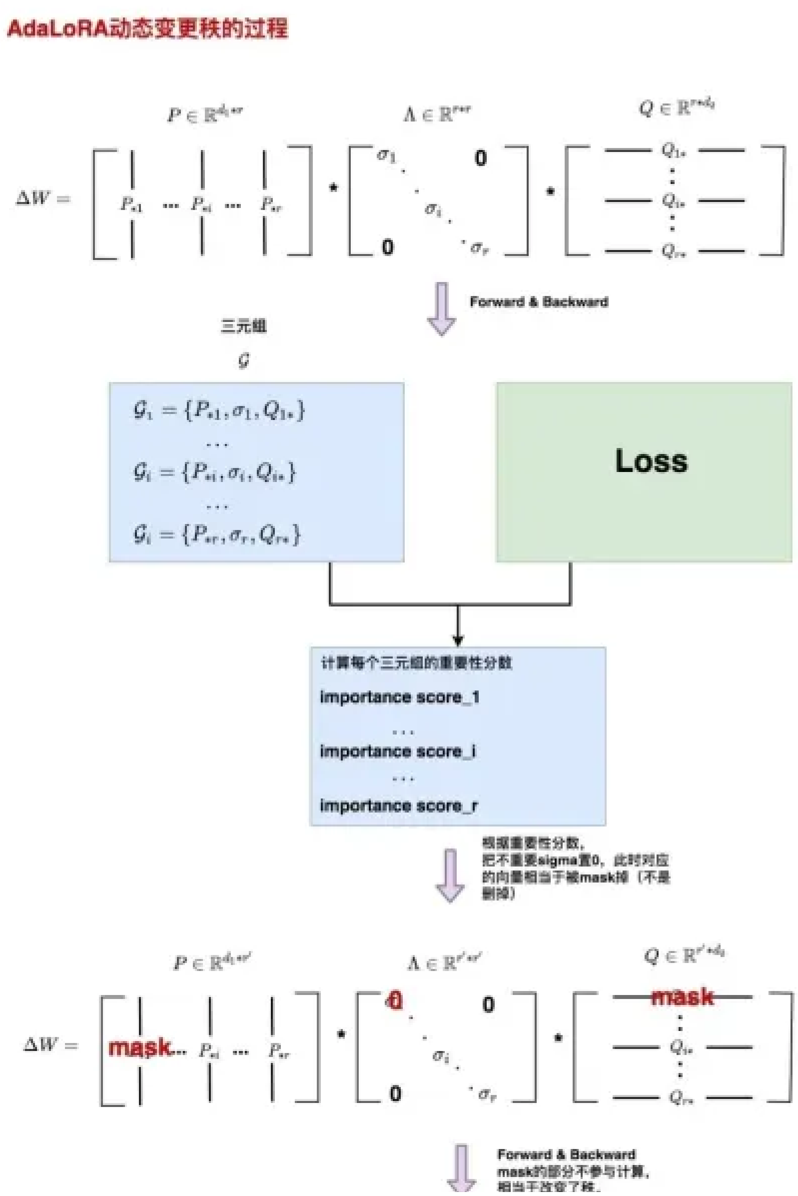
3. 具体流程

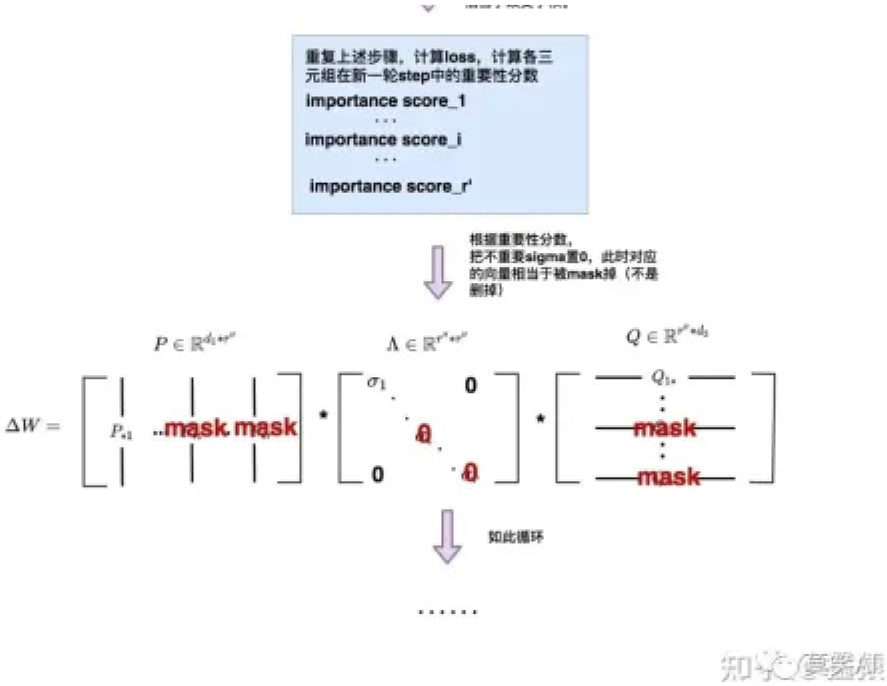
AdaLoRA变秩的整体流程如下：

1. **首先，初始化三个矩阵**P, Λ, Q 。其中 Λ 矩阵比较特殊，其大部分元素为0，只有对角线上的r个元素有值， $\Lambda \in \mathbb{R}^r$ 。初始化时，**将** Λ **初始化为0，** P和Q **初始化为高斯随机矩阵，** 这样做的目的为了在训练开始保证ΔW是0，以此避免引入噪声。
2. 然后，**正常做forward和backward，得到Loss和参数的梯度。**
3. 接着，**根据Loss和参数梯度**，对每个三元组(triplets) $\mathcal{G}_i\{P_{*i}, \sigma_i, Q_{i*}\}$ **计算重要性分数。**

P_{*i}, Q_{i*} 分别表示“第i列”和“第i行”。

4. 接着，**根据计算出来的重要性分数，将不重要的三元组挑选出来。**
5. 接着，**对于不重要的三元组，将σ'其值置0**。这样，在下一次做forward时，这个三元组里对应的P向量和Q向量**相当于**被mask掉了，对Loss没有贡献。也就起到了**变秩**的效果。
   a. **为什么只是将 σ' 置0，而不是把整个三元组删掉？**模型的学习是一个探索的过程，**在一开始模型认为不重要的三元组，在后续过程中模型可能会慢慢学到它的重要性。因此，mask是比删掉更合理的方法**。也正是这个原因，我们在步骤6中，不管三元组有没有被mask掉，我们都会正常用梯度更新P和Q。
6. 接着，使用 2 中计算出来的梯度，更新P和Q的参数。
7. 然后，使用更新完毕的 ，开启新一轮forward和backward，重复上面步骤，随时动态更新参数的秩。





4. 与LoRA对比

相比LORA，AdaLORA这种设计方式有两个优点：

- AdaLORA只裁剪奇异值矩阵，并不裁剪奇异向量，因此训练过程中更容易恢复被误删的奇异值。
- AdaLORA的P和Q正交矩阵，而LORA的A和B非正交。AdaLORA训练过程中裁剪操作不会影响其他奇异值对应的奇异向量，因此训练会更稳定泛化性能更好。**文档内容概述：AdaLoRA详解**