Adalora

lora存在的问题：

1、对所有模块都采用相同的秩

2、微调的过程中秩保持不变

针对这两个问题，Adalora做出了改进：

Adalora的整体目标是做**参数预算（parameter budget）**，也就是忽略不重要的参数，把训练资源给重要的参数。即模型在微调过程中，自己学习每个模块的参数对训练结果的重要性，然后，依据重要性，动态调整不同模块的秩。

LoRA

$$W = W^{(0)} + \Delta = W^{(0)} + BA,$$

$$\text{where } W^{(0)}, \Delta \in \mathbb{R}^{d_1 \times d_2}, A \in \mathbb{R}^{r \times d_2} \text{ and } B \in \mathbb{R}^{d_1 \times r} \text{ with } r \ll \{d_1, d_2\}.$$

AdaLoRA

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

$$P \in \mathbb{R}^{d_1 \times r} \text{ and } Q \in \mathbb{R}^{r \times d_2} \quad \Lambda \in \mathbb{R}^{r \times r}$$

为了保证P和Q为正交矩阵，在训练的时候需要添加如下正则项

$$R(P,Q) = \|P^\top P - I\|_\mathsf{F}^2 + \|QQ^\top - I\|_\mathsf{F}^2.$$

为什么要分解成P、Λ、Q三个矩阵，而不是像lora那样分解成B和A两个矩阵?

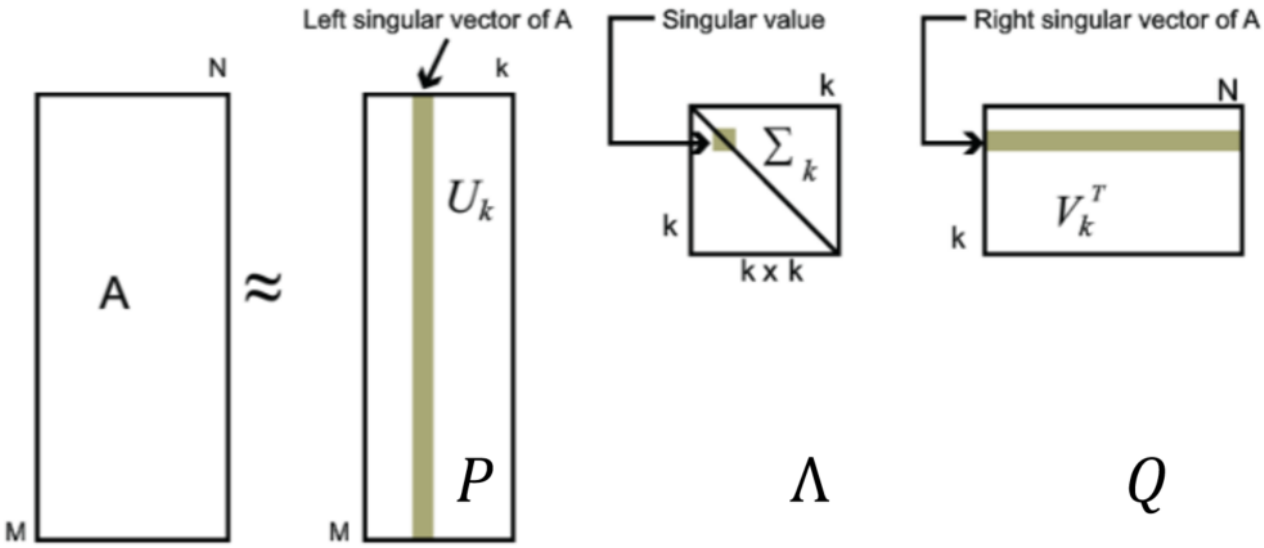
基于以下两点原因：

1、因为lora中不能保证A和B是正交的，也就是不能保证B的列向量和A的行向量是线性无关的，那么通过删除B的列向量和A的行向量来调节秩会对结果产生较大影响。

2、通过删除的这种方式来调节秩不够灵活，难以恢复。而adalora中通过把Λ矩阵中某个对角线元素置为0的方式可以灵活地更改秩。

怎么衡量哪个模块更重要呢?

importance score的概念



把图中绿色的部分称为一个三元组，三元组importance score计算方式如下：

$$S_{k,i} = s(\lambda_{k,i}) + \frac{1}{d_1}\sum_{j=1}^{d_1} s(P_{k,ji}) + \frac{1}{d_2}\sum_{j=1}^{d_2} s(Q_{k,ij}),$$

三元组的重要性分数 = λ的重要性分数 + P矩阵中所有元素重要性分数的均值 + Q矩阵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。

如何计算每个部分的importance score?

1. Magnitude of singular values 用Λ中的奇异值大小来衡量重要性，但效果不好
2. Sensitivity-based importance

$$S_{k,i} = s(\lambda_{k,i}) + \frac{1}{d_1}\sum_{j=1}^{d_1} s(P_{k,ji}) + \frac{1}{d_2}\sum_{j=1}^{d_2} s(Q_{k,ij}),$$

参考了模型可解释性中的方法

$$s(x) = x \cdot \nabla_x L \qquad L(x) = L(0) + x \cdot \frac{dL}{dx} + O(x^2)$$

梯度 loss 泰勒展开

但是在SGD的过程中，每次见到的数据是一个minibatch，对梯度的估计不太准确，因此考虑把不同batch中的结果做一个动量累加：

3. Sensitivity-based importance– Plus (moving average)

$$I(w_{ij}) = |w_{ij}\nabla_{w_{ij}}\mathcal{L}|,$$

可以理解为importance score的平均数：$\bar{I}^{(t)}(w_{ij}) = \beta_1 \bar{I}^{(t-1)}(w_{ij}) + (1-\beta_1) I^{(t)}(w_{ij})$

可以理解为importance score的方差：$\bar{U}^{(t)}(w_{ij}) = \beta_2 \bar{U}^{(t-1)}(w_{ij}) + (1-\beta_2)\left|I^{(t)}(w_{ij}) - \bar{I}^{(t)}(w_{ij})\right|,$

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

整体算法流程

来源：[大猿搬砖简记][https://mp.weixin.qq.com/s/b1PkQ9CHDlmSblG5DhSPrA]

Algorithm 1 AdaLoRA

Input: Dataset $\mathcal{D}$; total iterations T; budget schedule $\{b^{(t)}\}_{t=0}^{T}$; hyperparameters $\eta, \gamma, \beta_1, \beta_2$.
for $t = 1, \ldots, T$ **do**
  Sample a mini-batch from $\mathcal{D}$ and compute the gradient $\nabla\mathcal{L}(\mathcal{P}, \mathcal{E}, \mathcal{Q})$;
  Compute the sensitivity $I^{(t)}$ in (8) for every parameter in $\{\mathcal{P}, \mathcal{E}, \mathcal{Q}\}$;
  Update $\overline{I}^{(t)}$ as (9) and $\overline{U}^{(t)}$ as (10) for every parameter in $\{\mathcal{P}, \mathcal{E}, \mathcal{Q}\}$; 计算importance score
  Compute $S_{k,i}^{(t)}$ by (7), for $k = 1, \ldots, n$ and $i = 1, \ldots, r$;
  Update $P_k^{(t+1)} = P_k^{(t)} - \eta\nabla_{P_k}\mathcal{L}(\mathcal{P}, \mathcal{E}, \mathcal{Q})$ and $Q_k^{(t+1)} = Q_k^{(t)} - \eta\nabla_{Q_k}\mathcal{L}(\mathcal{P}, \mathcal{E}, \mathcal{Q})$;
  Update $\Lambda_k^{(t+1)} = \mathcal{T}(\Lambda_k^{(t)} - \eta\nabla_{\Lambda_k}\mathcal{L}(\mathcal{P}, \mathcal{E}, \mathcal{Q}), S_k^{(t)})$ given the budget $b^{(t)}$. 置0的操作
end for
Output: The fine-tuned parameters $\{\mathcal{P}^{(T)}, \mathcal{E}^{(T)}, \mathcal{Q}^{(T)}\}$.

- （1）拿到训练数据集，确定好总训练步长T。根据总步长T设计好top_b的warm-up策略，并设定好一系列超参，同时也把P, Λ, Q的初始化做好
- （2）进入某个step的迭代
- （3）给模型喂一份mini-batch，正常做forward和backward，计算loss和梯度
- （4）（5）对某一个三元组，我们先计算其中每个参数的重要性（单参数重要性）
- （6）根据单参数重要性，计算出整个三元组的重要性分数
- （7）使用（3）中计算好的梯度，正常更新矩阵P和Q
- （8）根据三元组重要性分数、动态调秩策略、top_b来判断要给哪些入置0，其对应的三元组中的P和Q向量相当于被mask掉，以此来实现动态调秩的目的。这番操作后，我们得到更新的矩阵Λ。然后将P, Λ, Q送入下一轮训练。

以此类推。

怎么动态地调整r?

top_b策略：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。**这个过程就和warm-up非常相似。**

AdaLoRA动态变更秩的过程

Forward & Backward

三元组

$\mathcal{G}$

$\mathcal{G}_1 = \{P_{*1}, \sigma_1, Q_{1*}\}$

...

$\mathcal{G}_i = \{P_{*i}, \sigma_i, Q_{i*}\}$

...

$\mathcal{G}_i = \{P_{*r}, \sigma_r, Q_{r*}\}$

Loss

计算每个三元组的重要性分数

importance score_1

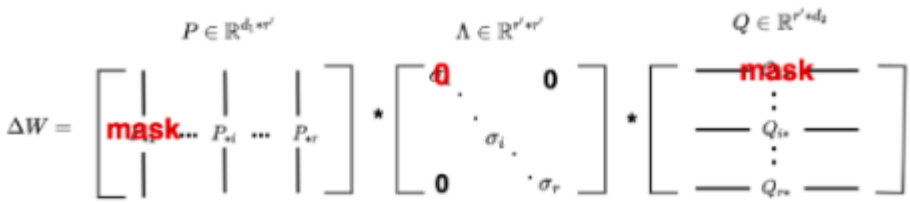
...

importance score_i

...

importance score_r

根据重要性分数，
把不重要sigma置0，此时对应的向量相当于被mask掉（不是删掉）



Forward & Backward
mask的部分不参与计算，
相当于改变了秩。

重复上述步骤，计算loss，计算各三元组在新一轮step中的重要性分数

importance score_1

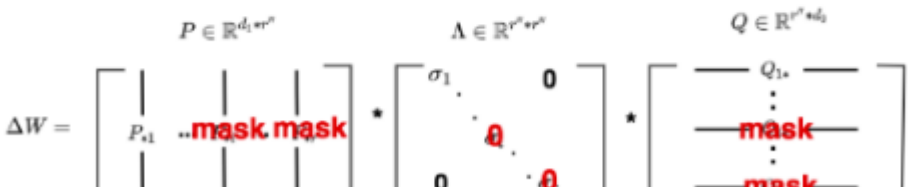
...

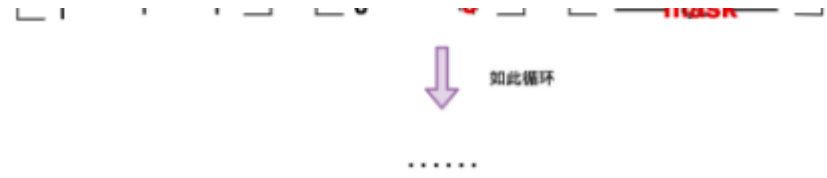
importance score_i

...

importance score_r'

根据重要性分数，
把不重要sigma置0，此时对应的向量相当于被mask掉（不是删掉）





.....

AdaLoRA变秩的整体流程如下：

（1）**首先，我们初始化三个矩阵**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。所以实操中我们可将其视为长度为r的向量，即$\Lambda \in \mathbb{R}^r$。

初始化时，我**们将**Λ**初始化为0，**P, Q**初始化为高斯随机矩阵。**这样做的目的和LoRA一样，都是为了在训练开始保证ΔW是0，以此避免引入噪声。

（2）**然后，我们正常做forward和backward**，得到Loss和参数的梯度。

（3）**根据Loss和参数梯度**，我们可以对图中所示的每个三元组(triplets)$\mathcal{G}_i\{P_{*i}, \sigma_i, Q_{i*}\}$**计算重要性分数**(importance scoring)，其中，P_{*i}, Q_{i*}分别表示"第i列"和"第i行"。

（4）**根据计算出来的重要性分数，我们将不重要的三元组挑选出来。**

（5）**对于不重要的三元组，我们将其**σ**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。

（6）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P, Λ, Q，开启新一轮forward和backward，重复上面步骤，随时动态更新参数的秩。

代码实现：

```
# ===== Before =====
# layer = nn.Linear(in_features, out_features)

# ===== After ======
import loralib
# Add a SVD-based adaptation matrices with rank r=12
layer = loralib.SVDLinear(in_features, out_features, r=12)

model = BigModel()
# This sets requires_grad to False for all parameters without the string "lora_" in their
names
loralib.mark_only_lora_as_trainable(model)

from loralib import RankAllocator
from loralib import compute_orth_regu
# Initialize the RankAllocator
rankallocator = RankAllocator(
    model, lora_r=12, target_rank=8,
    init_warmup=500, final_warmup=1500, mask_interval=10,
    total_step=3000, beta1=0.85, beta2=0.85,
)

# ===== Before =====
# loss.backward()
# optimizer.step()
# global_step += 1

# ===== After ======
(loss+compute_orth_regu(model, regu_weight=0.1)).backward
optimizer.step()
rankallocator.update_and_mask(model, global_step)
global_step += 1
```

Adalora的不足：

1、论文中公式过多，下标略显混乱

2、最大只在1B参数量的模型上做了实验