量化（5）：GPTQ的前世今生——OBD/OBS/OBC/GPTQ

之前系列的blog地址：

量化（1）：概念介绍

量化（2）：SmoothQuant

量化（3）：OmniQuant

量化（4）：AWQ

本 blog 学习 GPTQ的前世今生

Optimal Brain Damage（最优脑损伤）

一起读下这篇论文。论文中有些观点也是值得回味的。

> It is known from theory (Denker et al., 1987; Baum and Haussler, 1989; Solla et al., 1990) and experience (Le Cun, 1989) that, for a fixed amount of training data, networks with too many weights do not generalize well.

对于一定数量的训练数据，模型越大，泛化的能力就越差，所以大模型，除了模型参数量大，训练的数据也要足够大，不然模型的泛化能力可能就会差。

最好的泛化性能需要在训练误差和网络复杂性之间取得一个平衡。

> The best generalization is obtained by trading off the training error and the network complexity

一个方法就是在误差函数上加上一些衡量“模型复杂度”的东西

> One technique to reach this tradeoff is to minimize a cost function composed of two terms: the ordinary training error, plus some measure of the network complexity.

因此很多人提出了如何来衡量“模型的复杂度”，（这部分没有必要深究，比如最简单的就是用非零参数的个数来衡量）

> Various complexity measures have been proposed, including Vapnik-Chervonenkis dimensionality (Vapnik and Chervonenkis, 1971) and description length (Rissanen, A time-

> honored (albeit inexact) measure of complexity is simply the number of non-zero free parameters, which is the measure we choose to use in this paper [but see (Denker, Le Cun and Solla, 1990)]. Free parameters are used rather than connections, since in constrained networks, several connections can be controlled by a single parameter.

这篇文章的思路是，删除那些不重要的权重来减少模型的复杂度，这样才会对误差的影响最小，同时也减少了模型的复杂度，提高泛化能力。那哪些权重是不重要的呢，自然而然的想法就是数值越低的权重，重要性越小。

个人从神经元的角度来理解，神经网络就是模拟神经元，神经元之间的连接就是突触，突触之间的“连接强度”就是权重weight，权重越大，自然表示突触之间连接的越紧密。因此可以类推，权重越小，越不重要。

但是作者提出的方法，不是简单的通过直接衡量权重的大小来判断 “saliency(重要性)”，而是更有理论性的衡量。

思路也很简单：

一个 Linear 层：`F = W*X`

如果对 W 进行剪枝（减少模型参数），那么 `F' = W_ *X`，误差就是 `F - F'`，如果剪枝后，误差很小，说明 saliency 小，如果误差大，说明 saliency 越大。

这是定量和直觉上的描述，我们需要更精确的数学描述。

> Objective functions play a central role in this field; therefore it is more than reasonable to define the saliency of a parameter to be **the change in the objective function caused by deleting that parameter.**

但是，这种通过暂时删除某些参数，然后得到误差的变化，最后得到 “saliency”是费时又费力的，因此需要新的方法。

> It would be prohibitively laborious to evaluate the saliency directly from this definition, i.e. by temporarily deleting each parameter and reevaluating the objective function.

作者从误差函数开始分析，对误差函数进行泰勒展开：

A perturbation(扰动) δU of the parametervector will change the objective function by:

下面公式中，u_i 表示参数，δu_i 表示参数的变化，

δE 表示因为减枝带来的参数变化造成的整体误差的变化。

g_i 表示梯度

h_{ij} 表示参数对应的海瑟矩阵

下面公式就表示某个参数的变化造成的误差的变化。

$$\delta E = \sum_i g_i \delta u_i + \frac{1}{2} \sum_i h_{ii} \delta u_i^2 + \frac{1}{2} \sum_{i \neq j} h_{ij} \delta u_i \delta u_j + O(||\delta U||^3) \qquad (1)$$

Here, the δu_i's are the components of δU, the g_i's are the components of the gradient G of E with respect to U, and the h_{ij}'s are the elements of the Hessian matrix H of E with respect to U:

$$g_i = \frac{\partial E}{\partial u_i} \quad \text{and} \quad h_{ij} = \frac{\partial^2 E}{\partial u_i \partial u_j} \qquad (2)$$

因此，从公式中可以得到，计算权重 U 的梯度和海瑟矩阵，就知道误差了，但是 Hessian 矩阵的计算量非常之大。

假如权重有 2600 个参数，那么Hessian 矩阵有 2600 * 2600 项。难以计算。

> The goal is to find a set of parameters whose deletion will cause the least increase of E . This problem is practically insoluble in the general case. One reason is that the matrix H is enormous (6.5 x 10 6 terms for our 2600 parameter network), and is very difficult to compute.

所以近似简化是必要的：

1. 参数独立：认为参数之间都是独立的，所以二阶偏导数的交叉项可以认为是0， 忽略
2. 局部极值：认为训练收敛后，误差处于极值点位置，所以一阶导数为0，忽略。同时，在极值点附近，误差函数是凸函数，因此海瑟矩阵的对角线元素都是非负数，因此任何扰动都只会增加或者不改变误差。
3. 二次近似假定：忽略泰勒高阶项，只保留二次项

经过简化后只剩下了第二项，只需要计算H矩阵的对角项。它可以基于优化目标对连接权重的导数进行计算，复杂度就与梯度计算相同了，如下：

$$\delta E = \frac{1}{2} \sum_i h_{ii} \delta u_i^2$$

接下来的任务就是如何计算海瑟矩阵的对角线元素，经过一顿推导和近似（具体过程看论文）：

$$\frac{\partial^2 E}{\partial a_{ii}^2} = 2 f'(a_i)^2 - 2(d_i - x_i) f''(a_i)$$

第二项可以近似忽略，因此得出结论，直接从一阶偏导数就可以得到海瑟矩阵的对角线元素，这样就不会引入额外的运算。

到这里，重要性 **saliency** 定量的计算方式就是：

$$\delta E = \frac{1}{2}\sum_i h_{ii}\delta u_i^2$$

对所有的参数，计算它的二阶导数，就知道了参数的 saliency，然后进行重要性排序，删除那些 low-saliency 的权重。然后重新训练使模型收敛，继续重续上述流程。

整个剪枝过程如下：

The OBD procedure can be carried out as follows:

1. Choose a reasonable network architecture
2. Train the network until a reasonable solution is obtained
3. Compute the second derivatives h_{kk} for each parameter
4. Compute the saliencies for each parameter: $s_k = h_{kk}u_k^2/2$
5. Sort the parameters by saliency and delete some low-saliency parameters
6. Iterate to step 2

Optimal Brain Surgeon （最优脑手术）

简单过下这篇论文的思想！

前面的 OBD，在考虑误差的时候，只保留了海瑟矩阵的对角线元素，即作者在对误差函数的简化过程中使用了参数独立假设，但实际上，参数之间会相互影响，因此，OBS考虑了海瑟矩阵的所有元素来计算误差。

下面这幅图可以解释 OBS 的思想：

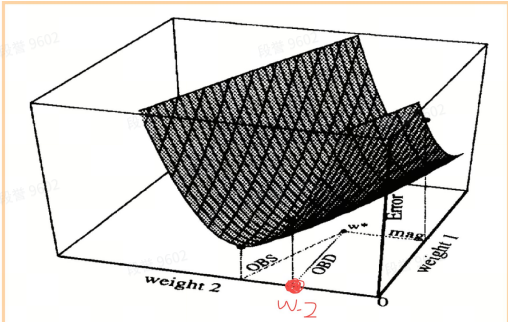


Figure 1: Error as a function of two weights in a network. The (local) minimum occurs at weight w*, found by gradient descent or other learning method. In this illustration, a magnitude based pruning technique (mag) then removes the smallest weight, weight 2; Optimal Brain Damage (OBD) before retraining removes weight 1. In contrast, our Optimal Brain Surgeon method (OBS) not only removes weight 1, but also automatically adjusts the value of weight 2 to minimize the error, without retraining. The error surface here is general in that it has different curvature (second derivatives) along different directions, a minimum at a non-special weight value, and a non-diagonal Hessian (i.e., principal axes are not parallel to the weight axes). We have found (to our surprise) that every problem we have investigated has strongly non-diagonal Hessians - thereby explaining the improvement of our method over that of Le Cun et al.

注意，实际origin weight 2 的位置在图中红色点的位置，weight_1 > w_2

那么根据经典的基于weight值大小的优化方法：w_2小一点，所以重要性更小，所以会 remove w2。

根据 OBD 的思想，先需要计算误差曲面在点 w_2 和 weight_1 上的二阶偏导数（二阶偏导的几何意义是曲面沿某个轴线的切线的变化率），从而判断重要性，（这里定量分析下，不做准确的计算）误差表面在 w_2 附近沿 w_2 所在轴的切线变化率大于误差表面在 weight_1 附近沿 weight_1 所在轴的切线变化率，因此 w_2 更重要，所以会 remove weight_1。

OBS 还要考虑二阶混合偏导（二阶混合偏导的几何意义我也不是很明白），同样也是 w_2 的saliency 更大，因此remove weight_1，但同时，OBS还会对 w_2 修正，w_2 修正到 weight_2, 从图中也可以直观的看出，修正后，weight_2 的误差显著小于 w_2，并且 weight_2 的误差接近于 w* 的误差。

下面展示具体的理论推导过程。

和OBD类似的分析，误差展开为泰勒级数，然后同时省去一次项和高阶项，

$$\delta E = \frac{1}{2}\delta W^T * H * \delta W \tag{1}$$

公式（1）表示，权重 W 的变化引起的误差的变化。

剪枝的目的是去掉权重 W 中的某些参数以达到缩减参数量的目的，这句话可以表达为约束：

$$e_q^T \delta W + w_q = 0 \tag{2}$$

$e_q \in (n, 1)$ 是单位列向量

$e_q^T \in (1, n)$ 是单位行向量

$\delta W \in (n,1)$ 是一个列向量，n 表示参数的总个数

w_q 表示一个 scalar

面对带有约束的优化问题，可以通过引入拉格朗日乘子构造拉格朗日函数，转换为无约束问题。

$$L = \frac{1}{2}\delta W^T * H * \delta W + \lambda(e_q^T \delta W + w_q) \tag{3}$$

接下来求 L 对 δW 的偏导，并令其为零（ λ 是拉格朗日算子，是个 scalar， w_q 也是 scalar，表示被剪枝的某个权重）：

$$\frac{\partial L}{\partial \delta W} = H * \delta W + \lambda e_q = 0$$

$H \in (n,n)$， $\delta W \in (n,1)$， $e_q \in (n,1)$，得到：

$$\delta W = -H^{-1} * \lambda e_q \tag{4}$$

$$\delta W_q = -e_q^T * H^{-1} * \lambda e_q = -[H^{-1}]_{qq}\lambda$$

$[H^{-1}]_{qq}$ 表示海瑟矩阵对角线上qq位置的数值，是一个 scalar，再根据公式（2），得到：

$$\lambda = -\frac{\delta W_q}{[H^{-1}]_{qq}} = \frac{w_q}{[H^{-1}]_{qq}} \tag{5}$$

把公式（4）代入公式（1）得：

$$\delta E = \frac{1}{2}(-H^{-1} * \lambda e_q)^T * H * (-H^{-1} * \lambda e_q) = \frac{1}{2}\lambda^2 e_q^T H^{-1} H H^{-1} e_q = \frac{1}{2}\lambda^2 e_q^T H^{-1} e_q = \frac{1}{2}\lambda^2 [H^{-1}]_{qq}$$

再把公式（5）代入：

$$\delta E = \frac{1}{2}\frac{w_q^2}{[H^{-1}]_{qq}} \tag{6}$$

公式（6）意义重大：它表示了剪枝掉一个权重 w_q 之后，引起的误差变化的大小，即 **saliency**。

不妨比较一下这个 **saliency 计算公式和 OBD 中的 saliency 计算公式的区别，主要区别是一个直接计算H，另一个需要计算H的逆。**

根据这个公式，只需要计算出海瑟矩阵的逆，然后根据对角线上的元素，便可以判断出权重中每个参数的重要程度（saliency），然后排序，去掉那些最不重要的权重。

把公式（5）代入公式（4）：

$$\delta W = -\frac{w_q}{[H^{-1}]_{qq}}[H^{-1}]e_q = -\frac{w_q}{[H^{-1}]_{qq}}[H^{-1}]_{:,q} \tag{7}$$

公式（7）意义重大：**它表示剪枝掉某一个参数时，对其他 n-1 个权重的补偿（为了使误差最小化的一个补偿）。就这完美呼应了对开篇图的理解。**

因此，OPS的整体流程是：

Table 1: *Optimal Brain Surgeon procedure*

1. **Train a "reasonably large" network to minimum error.**
2. **Compute $\mathbf{H}^{-1}$.**
3. **Find the q that gives the smallest saliency $L_q = w_q^2/(2[\mathbf{H}^{-1}]qq)$. If this candidate error increase is much smaller than E, then the qth weight should be deleted, and we proceed to step 4; otherwise go to step 5. (Other stopping criteria can be used too.)**
4. **Use the q from step 3 to update *all* weights (Eq. 5). Go to step 2.**
5. **No more weights can be deleted without large increase in E. (At this point it may be desirable to retrain the network.)**

步骤3中的 saliency 就是推导的公式（6），如果 w_q 权重引起的误差的增加远远小于当前的误差 E，就可以考虑删掉该权重，然后对其他参数进行补偿。流程上与 OBD 最大的区别是不需要重新训练了，这极大节省了计算资源。

Optimal Brain Compression（剪枝和量化都是模型压缩）

OBC 主要解决 OBS 中计算量大的问题。

OBS对整个神经网络进行剪枝，OBC对神经网络模型分层剪枝或者量化。

上面OBS的的流程中，每一步先计算每个权重的 saliency，然后剪掉 saliency 最小的权重，接着对其他未剪枝的权重进行补偿。

优化目标依然是：

$$argmin \mid\mid WX - \hat{W}X \mid\mid_2^2 \tag{8}$$

$\hat{W}$ 表示量化后的权重，根据前面的 OBS 分析结论，剪切一个权重 w_q 造成的误差和需要的补偿如下：

$$\delta E = \frac{1}{2}\frac{\delta W^2}{[H^{-1}]_{qq}} = \frac{1}{2}\frac{(0-w_q)^2}{[H^{-1}]_{qq}} = \frac{1}{2}\frac{w_q^2}{[H^{-1}]_{qq}} \tag{9}$$

$$\delta W = \frac{\delta W}{[H^{-1}]_{qq}}[H^{-1}]e_q = \frac{0-w_q}{[H^{-1}]_{qq}}[H^{-1}]e_q = -\frac{w_q}{[H^{-1}]_{qq}}[H^{-1}]e_q = -\frac{w_q}{[H^{-1}]_{qq}}[H^{-1}]_{:,q} \tag{10}$$

该过程中，计算海瑟矩阵的逆时，需要的计算量是 $O(d^3)$， $d = d_{row} * d_{col}$ ，然后迭代流程需要遍历所有的权重，算法的复杂度为 $O(d^4)$

The ExactOBS Algorithm

OBS 提出了一种优化算法（row-wise）。作者观察到，删除一个权重只会影响结果中的对应行，故可以认为行与行之间参数的压缩是独立的。优化的目标因此可以改写为：

$$\sum_{i=1}^{d_{row}} || W_{i,:}X - \hat{W}_{i,:}X ||_2^2 \tag{11}$$

对于每一行来说： $Y_{i,:} = W_{i,:}X$ ，优化 $argmin || W_{i,:}X - \hat{W}_{i,:}X ||_2^2$ 就是一个标准的二次型问题。海瑟矩阵 H 的大小缩减为了 (d_{col}, d_{col}) ，二次型问题 H 的计算方式为（直接对优化目前函数求导可得）： $H = 2XX^T$,然后求逆，整体的计算量缩减为 $O(d_{row} * d_{col}^3)$ 。对每一行来说，剪枝流程如下：

Algorithm 1 Prune $k \leq d_{\text{col}}$ weights from row $\mathbf{w}$ with inverse Hessian $\mathbf{H}^{-1} = (2\mathbf{X}\mathbf{X}^\top)^{-1}$ according to OBS in $O(k \cdot d_{\text{col}}^2)$ time.

$M = \{1, \ldots, d_{\text{col}}\}$
for $i = 1, \ldots, k$ **do**
  $p \leftarrow \text{argmin}_{p \in M} \frac{1}{[\mathbf{H}^{-1}]_{pp}} \cdot w_p^2$
  $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}^{-1}_{:,p} \frac{1}{[\mathbf{H}^{-1}]_{pp}} \cdot w_p$
  $\mathbf{H}^{-1} \leftarrow \mathbf{H}^{-1} - \frac{1}{[\mathbf{H}^{-1}]_{pp}} \mathbf{H}^{-1}_{:,p} \mathbf{H}^{-1}_{p,:}$
  $M \leftarrow M - \{p\}$
end for

1. 根据公式（9）计算该行 col 个参数的 saliency，然后排序，得到最小的 saliency 对应的参数（贪心策略）
2. 计算补偿，补偿其他不剪枝的参数
3. 更新海瑟矩阵的逆

这里对这一步展开讨论一下：首先说明下论文里的这个图稍微有点瑕疵。

为什么要更新海瑟矩阵的逆？因为在剪枝了某个参数后，原本的二次型优化问题总的参数从 n 变成了 n-1 ，假设是第p个参数被剪枝了，剪枝后的优化目标变成了

$$argmin || w_{:,-p}X_{-p,:} - \hat{w_{:,-p}}X_{-p,:} ||_2^2$$

w 表示原 W 中的某一行， $w_{:,-p}$ 表示去掉第p个参数后的权重，$X_{-p,:}$ 表示 X 中的去掉第 p 行，重新计算 H:

$$new\_H = 2X_{-p:,}X_{-p:,}^T = H_{-p}$$

很容易判断出，新的 H 就等于原来的 H 去掉第p行第p列（ H_{-p} ）。根据高斯消元法，它的逆计算方式如下：

$$\mathbf{H}_{-p}^{-1} = \left(\mathbf{H}^{-1} - \frac{1}{[\mathbf{H}^{-1}]_{pp}}\mathbf{H}_{:,p}^{-1}\mathbf{H}_{p,:}^{-1}\right)_{-p},$$

这是论文中给的公式，但是容易引起误解，更好的写法如下：

$$[H_{-p}]^{-1} = (H^{-1} - \frac{1}{[H^{-1}]_{pp}}[H^{-1}]_{:,p}[H^{-1}]_{p,:})_{-p}$$

$-p$ 尾缀表示原来的矩阵去掉第p行第p列， 因此，论文给出的流程中，用这个公式更好一些。

4. 完成对该参数的剪枝操作

(OBQ)量化和剪枝一样，根据公式（9）和（10）可以类比出量化时的 saliency 计算公式以及其他未量化权重的更新公式：

$$w_p = \text{argmin}_{w_p}\frac{(\text{quant}(w_p) - w_p)^2}{[\mathbf{H}^{-1}]_{pp}}, \quad \boldsymbol{\delta_p} = -\frac{w_p - \text{quant}(w_p)}{[\mathbf{H}^{-1}]_{pp}} \cdot \mathbf{H}_{:,p}^{-1}.$$

公式中的 $H_{:,p}^{-1} = [H^{-1}]_{:,p}$ ，表示H的逆中的第p列

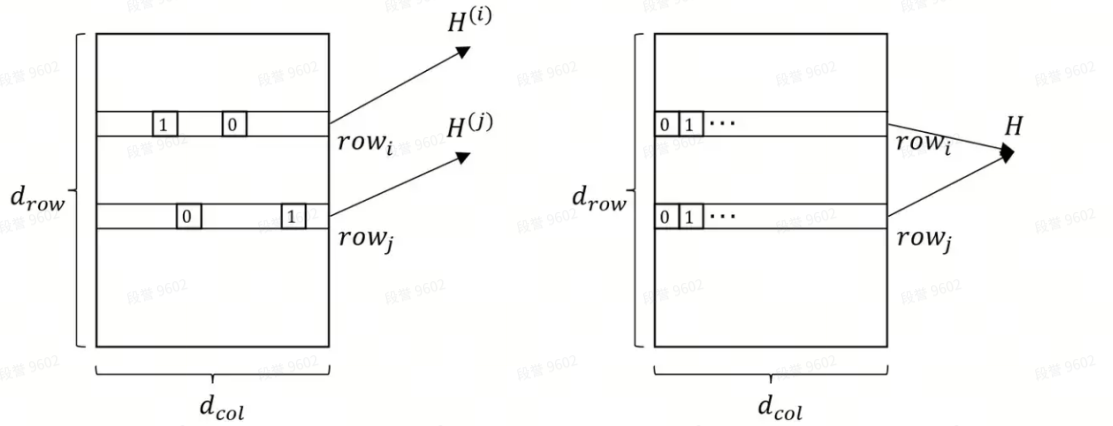
GPTQ（结合代码分析）

OBQ 采用了 row-wise 的方法，GPTQ 作者认为这种贪心策略虽然能达很高的准确性，但相对于任意顺序的权重选择方式来说提升效果并不是很大，如果让所有行都按一样的顺序来量化权重，则可以极大的简化量化过程，进一步降低计算量。

GPTQ 的改进点主要如下：

1. 采用固定顺序同时量化所有行

这里借用知乎上一博主 sudit 的画的图：



左边就是 OBC 的贪心策略，逐行根据 saliency 排序来决定量化的顺序。而右边是 GPTQ 提出的，对于每一行，按照统一固定的顺序来量化，这样的好处是什么呢？

对于权重矩阵的每一行来说，其初始 Hessian 矩阵只与输入矩阵 ☒ 相关，

$$H = 2XX^T$$

因此都是相同的。在 OBQ 中，每一行权重的优化顺序都可能不一样，因此每一行对应的 Hessian 矩阵也都会变得不一样，需要单独保存。而当所有行都按一样的顺序来量化权重时，**每一行对应的 Hessian 矩阵都是相同的**，因此逆矩阵也相同，这就意味着只需要计算 ☒☒☒☒ 次逆矩阵，而不是 ☒☒☒☒×☒☒☒☒ 次，这样一来总的时间复杂度就降低到了 ☒(☒☒☒☒3)，同时也省略了公式 (9) 寻找最优权重的计算过程。同时，补偿公式调整为如下：

$$\delta W = -\frac{w_q - quant(w_q)}{[H^{-1}]_{qq}}[H^{-1}]_{:,q} = -\frac{w_q - quant(w_q)}{[H^{-1}_{q:,q:}]_{0,0}}[H^{-1}_{q:,q:}]_{:,0} \tag{12}$$

再考虑到所有行都是按照一样的顺序来量化权重，那么一次迭代就可以量化同一列的所有行，于是我们可以构造向量化的权重调整公式：

$$\delta W = -\frac{W_{:,q} - quant(W_{:,q})}{[H^{-1}_{q:,q:}]_{0,0}}[H^{-1}_{q:,q:}]_{:,0} \tag{13}$$

其中 $W_{:,q}$ 表示权重矩阵的第 ☒ 列。同时 H 的更新公式可以更新为：

先摆出原来的 H 的更新公式：（再次强调，$-p$ 尾缀表示原来的矩阵去掉第p行第p列）

$$[H_{-p}]^{-1} = (H^{-1} - \frac{1}{[H^{-1}]_{pp}}[H^{-1}]_{:,p}[H^{-1}]_{p,:})_{-p}$$

更新后：

$$[H_{q:,q:}]^{-1} = ([H_{q-1:,q-1:}]^{-1} - \frac{1}{[[H_{q-1:,q-1:}]^{-1}]_{00}}[[H_{q-1:,q-1:}]^{-1}]_{:,0}[H_{q-1:,q-1:}]^{-1}]_{0,:})_{1:,1:} \tag{14}$$

对比这看，就容易理解了。

这里先停一下，一起看看代码，
https://github.com/AutoGPTQ/AutoGPTQ/blob/main/auto_gptq/quantization/gptq.py

GPTQ 中，主要的类是 <class GPTQ>，主要有两个方法 `add_batch`， `fasterquant`，`add_batch` 是用来计算**初始海瑟矩阵**的，对于每一行来说，初始海瑟矩阵的shape 为 `（d_col，d_col)`，代码如下（下面简化了原始代码，只突出重点部分）：

```
class GPTQ:
    def __init__(self, layer):
        self.layer = layer
        self.dev = self.layer.weight.device
        W = layer.weight.data.clone()
        self.rows = W.shape[0]
        self.columns = W.shape[1]
        self.H = torch.zeros((self.columns, self.columns), device=self.dev) # 初始海瑟矩阵的shape为（d_col, d_col)
        self.nsamples = 0

    def add_batch(self, inp, out): # inp: (batch, seq_len, hidden_size)
        tmp = inp.shape[0] # get batch
        self.H *= self.nsamples / (self.nsamples + tmp)
        self.nsamples += tmp
        # inp = inp.float()
        inp = math.sqrt(2 / self.nsamples) * inp.float()
        # self.H += 2 / self.nsamples * inp.matmul(inp.t())
        self.H += inp.matmul(inp.t()) # x*xT 计算第 i 个 输入的海瑟矩阵
```

前面说到初始海瑟矩阵的计算 $H = 2XX^T$ ，这仅针对于输入只有一个时，量化时，校准数据时有多条输出，所以优化目标实际上是：

$$argmin \frac{\sum_{i=1}^{n} || WX_i - \hat{W}X_i ||_2^2}{N}$$

对应的初始海瑟矩阵为：

$$H = \frac{\sum_{i=1}^{n} 2X_i X_i^T}{N}$$

计算时，如果一次性计算，可能导致显存不够，因此采用了迭代的方式计算，根据 `add_batch` 源码的计算方式，不妨写出前面几个 H_i 的计算方式：

$$H_1 = \sqrt{2}X_1 * \sqrt{2}X_1^T = 2X_1X_1^T$$

$$H_2 = \frac{H_1}{2} + \sqrt{\frac{2}{2}} X_2 * \sqrt{\frac{2}{2}} X_2^T = \frac{H_1}{2} + X_2 X_2^T = \frac{2X_1 X_1^T + 2X_2 X_2^T}{2}$$

$$H_3 = \frac{2H_2}{3} + \sqrt{\frac{2}{3}} X_3 * \sqrt{\frac{2}{3}} X_3^T = \frac{2H_2}{3} + \frac{2X_3 X_3^T}{3} = \frac{2X_1 X_1^T + 2X_2 X_2^T + 2X_3 X_3^T}{3}$$

可以看出 `add_batch` 源码就是迭代式的计算上述初始海瑟矩阵。

2. Cholesky分解

应用 Cholesky 分解来解决 H 逆矩阵计算的数值稳定性问题，更重要的是，简化了更新 H 的计算方式，要是每一步都根据公式（14）来更新海瑟矩阵，还是太麻烦了。更新的原理如下。

> Cholesky 分解很简单: 把一个对称正定的矩阵表示成一个下三角矩阵L和其转置的乘积的分解。

$H = LL^T$ 或者分解为上三角矩阵 $H = U^T U$

求的 H^{-1} 后，我们对它进行分解得到：

$$H^{-1} = U^T U$$

$$H = U^{-1}[U^T]^{-1}$$

注意： U^{-1} 是一个上三角，它的转置就是一个下三角，一个上三角和一个下三角矩阵相乘有如下性质：

$$H_{q:,q:} = [U^{-1}]_{q:,q:}[[U^T]^{-1}]_{q:,q:}$$

读者可以自己画一画，体会一下这个性质。然后两个继续求逆：

$$[H_{q:,q:}]^{-1} = [[[U^T]^{-1}]_{q:,q:}]^{-1} * [[U^{-1}]_{q:,q:}]^{-1}$$

它等同于(根据分块矩阵逆的性质，读者也可以举一些简单的例子来验证)：

$$[H_{q:,q:}]^{-1} = [U^T]_{q:,q:} * U_{q:,q:}$$

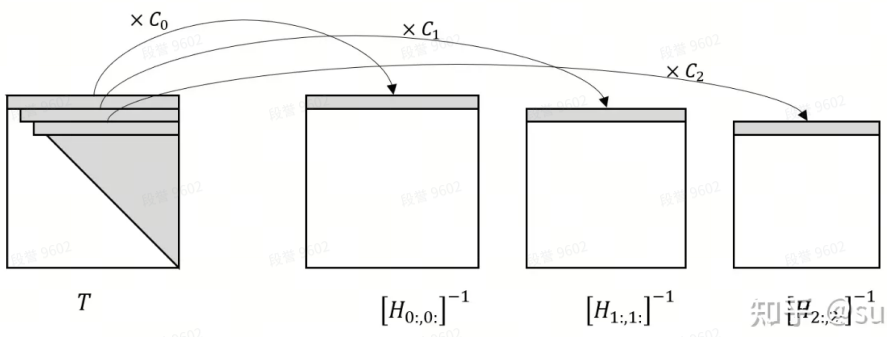
此时， U^T 是一个下三角矩阵，很明显有（读者可以画示意图演示）：

$$[[H_{q:,q:}]^{-1}]_{0,:} = C_q * [U_{q:,q:}]_{0,:} = C_q * U_{q,q:}$$

$$[[H_{q:,q:}]^{-1}]_{0,0} = C_q * [U_{q:,q:}]_{0,0} = C_q * U_{q,q}$$

海瑟矩阵是对称矩阵，它的逆也是对称的，即： $[[H_{q:,q:}]^{-1}]_{0,:} = [[H_{q:,q:}]^{-1}]_{:,0}$

这里再次引用博主 sudit 的画的图：



图中的 T 就是上述说的 Cholesky 分解 H^{-1} 得到的 U！

至此，我们不必再使用公式（14）来更新海瑟矩阵的逆，权重调整公式（13）也等价于：

$$\delta W = -\frac{W_{:,q} - quant(W_{:,q})}{[H^{-1}_{q:,q:}]_{0,0}}[H^{-1}_{q:,q:}]_{:,0} = -\frac{W_{:,q} - quant(W_{:,q})}{C_q U_{q,q}} C_q U_{q,q:} \tag{15}$$

$$\delta W = -\frac{W_{:,q} - quant(W_{:,q})}{U_{q,q}} U_{q,q:} \tag{16}$$

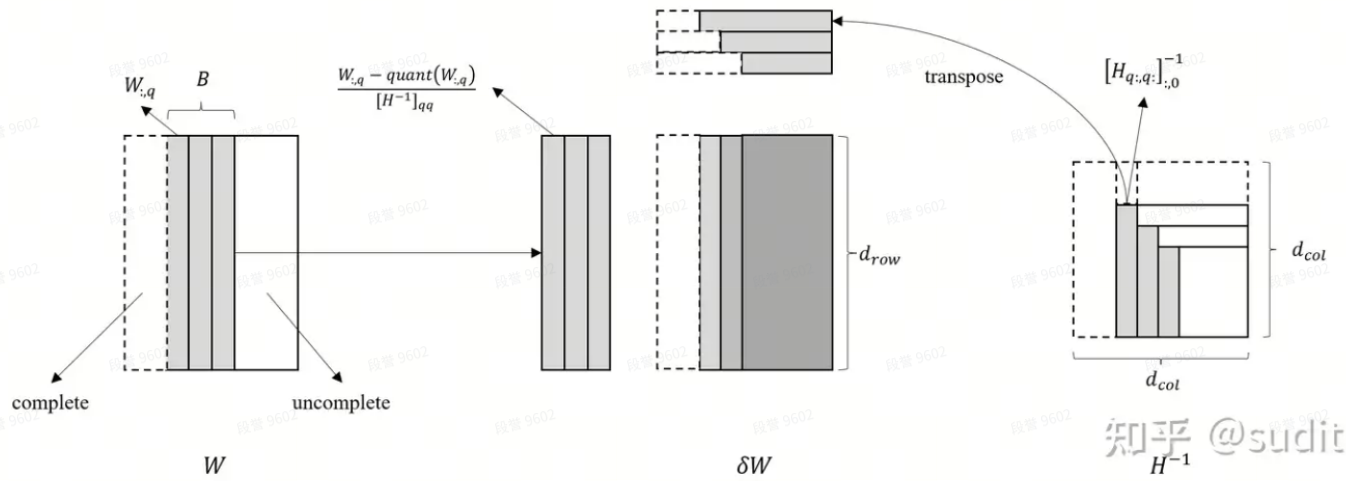
至此，我们只需要对初始的 H^{-1} 做一次分解得到一个上三角矩阵 U ，不必在每次迭代过程中更新海瑟矩阵的逆了，大幅减少了计算量！

3. 延迟批量更新

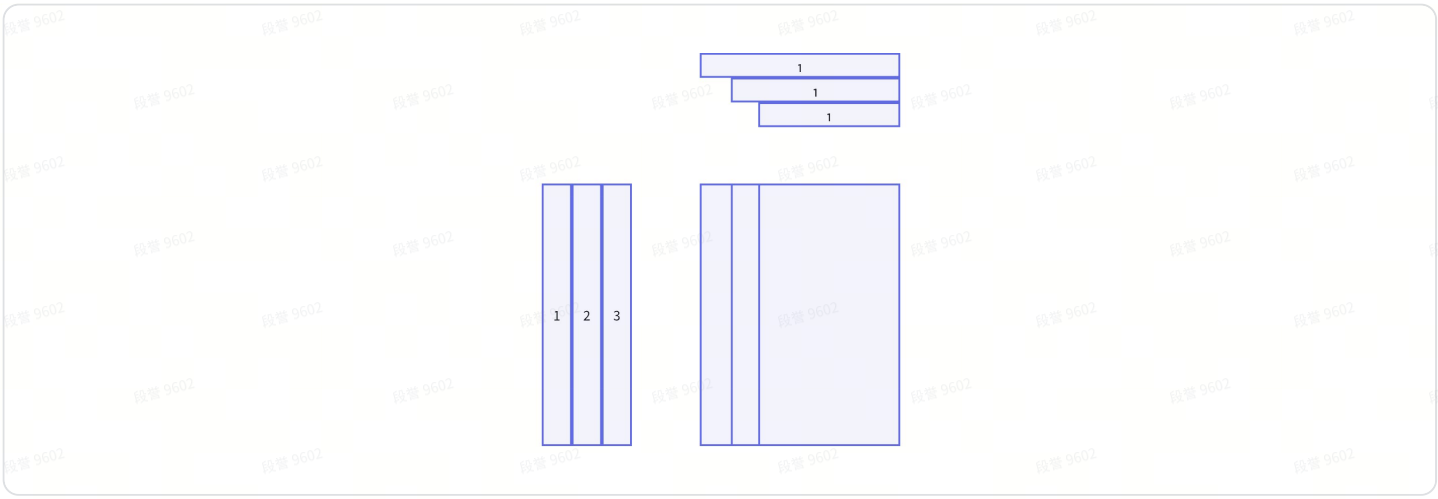
有些说法叫做“批量量化”其实是不太对的，按照论文里的说法应该叫“延迟批量更新”。本质就是为了充分发挥GPU的算力，优化权重跟新公式（16）的计算

量化还是按照固定的顺序一列一列的来，但是在计算补偿的时候，有一些优化技巧，目的就是为了充分发挥 GPU 的算力。

一列一列的量化流程如下：



在量化每一列权重时（一列一列的量化），需要和 $[[H_{q:,q:}]^{-1}]_{:,0}$ 即 $U_{q,q:}$ 做矩阵乘法计算补偿，$U_{q,q:}$ 它是阶梯型减小的，正如图所示。量化一列，补偿剩余未量化的所有列，继续量化第二列，然后再补偿剩余未量化的所有列，依次进行。关键是每次计算补偿的时候，是一个列向量，乘以一个行向量，浪费了GPU 的算力。



这个流程可以等价于下列流程：

以图中的假设每3列组成一个 block，量化第1列时，只计算第第二列和第三列的补偿，补偿第二列和第三列，然后量化第二列，再计算第三