llama.cpp

Agha Durrani

August 2024

1 `static float make_qkx2_quants`

Given a vector of weights $\mathbf{X} = [x_1, x_2, x_3, \ldots, x_n]$, where each weight x_i is a real number, and an unsigned quantization range of b bits, the process of asymmetric quantization can be described as follows:

Given x_{min} and x_{max}, which are simply:

$$x_{\min} = \min(\mathbf{X})$$

$$x_{\max} = \max(\mathbf{X})$$

and adjusting the minimum value if its greater than 0, by setting it equal to zero to avoid inefficient usage of the quantized range:

$$x_{\min} = 0 \text{ if } x_{\min} > 0$$

We can get initial estimate for the optimal scaling parameter, denoted as iscale and scale which is obtained through adopting uniform quantization, calculated using the formula:

$$\text{iscale} = \frac{2^b - 1}{x_{\max} - x_{\min}}$$

$$\text{scale} = \frac{1}{\text{iscale}}$$

Hence, quantized values l for weights x_i are computed as follows:

$$l_i = \text{nearest\_int}(\text{iscale}(\cdot x_i - x_{\min}))$$

To ensure that quantized values stay within their designated range, they are clipped according to their specified bounds:

$$max(0, min(2^b - 1, l_i)$$

To estimate the loss of information, one can easily determine the error by calculating the reconstruction value dq_i:

$$\text{dq}_i = \text{scale} \cdot L[i] + x_{\min}$$

The squared error for each weight is:

$$\text{error}_i = (\text{dq}_i - x_i)^2$$

The average mean squared error for the entire vector can then be expressed as:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (\text{scale} \cdot l_i + x_{\min} - x_i)^2$$

This initial value for MSE can now serve as a baseline for further improvement.

To find more optimal values for scale and $x_{\min}$, one can compute the partial derivatives of MSE with

respect to scale and min and set them to 0.

the partial derivative with respect to scale can be defined as:

$$\frac{\partial \text{MSE}}{\partial \text{scale}} = \frac{2}{n}\sum_{i=1}^{n}(\text{scale} \cdot l_i + x_{\min} - x_i) \cdot l_i$$

setting this to zero yields:

$$\frac{2}{n}\sum_{i=1}^{n}(\text{scale} \cdot l_i + x_{\min} - x_i) \cdot l_i = 0$$

which can be simplified to:

$$\sum_{i=1}^{n}(\text{scale} \cdot l_i + x_{\min} - x_i) \cdot l_i = 0$$

while the partial derivative with respect to $x_{\min}$ can be written as:

$$\frac{\partial \text{MSE}}{\partial x_{\min}} = \frac{2}{n}\sum_{i=1}^{n}(\text{scale} \cdot l_i + x_{\min} - x_i)$$

setting this to zero yields:

$$\frac{2}{n}\sum_{i=1}^{n}(\text{scale} \cdot l_i + x_{\min} - x_i) = 0$$

which can be simplified to:

$$\sum_{i=1}^{n}(\text{scale} \cdot l_i + x_{\min} - x_i) = 0$$

we can now represent these equations in a matrix form:

$$\begin{bmatrix} \sum_{i=1}^{n} l_i^2 & \sum_{i=1}^{n} l_i \\ \sum_{i=1}^{n} l_i & n \end{bmatrix} \begin{bmatrix} \text{scale} \\ x_{\min} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} x_i \cdot l_i \\ \sum_{i=1}^{n} x_i \end{bmatrix}$$

lets define matrix D as:

$$D = \begin{bmatrix} \sum_{i=1}^{n} l_i^2 & \sum_{i=1}^{n} l_i \\ \sum_{i=1}^{n} l_i & n \end{bmatrix}$$

If $\det(D) > 0$, then a unique and well-behaved solution exists. The optimal values for scale and $x_{\min}$ can then be determined by solving the following:

$$\begin{bmatrix} \text{scale} \\ x_{\min} \end{bmatrix} = D^{-1} \begin{bmatrix} \sum_{i=1}^{n} x_i \cdot l_i \\ \sum_{i=1}^{n} x_i \end{bmatrix}$$

The inverse of matrix D is:

$$D^{-1} = \frac{1}{\det(D)} \begin{bmatrix} n & -\sum_{i=1}^{n} l_i \\ -\sum_{i=1}^{n} l_i & \sum_{i=1}^{n} l_i^2 \end{bmatrix}$$

Thus, the optimal scale and $x_{\min}$ can be computed as:

$$\begin{bmatrix} \text{scale} \\ x_{\min} \end{bmatrix} = \frac{1}{\det(D)} \begin{bmatrix} n & -\sum_{i=1}^{n} l_i \\ -\sum_{i=1}^{n} l_i & \sum_{i=1}^{n} l_i^2 \end{bmatrix} \begin{bmatrix} \sum_{i=1}^{n} x_i \cdot l_i \\ \sum_{i=1}^{n} x_i \end{bmatrix}$$

However, since we require $x_{\min} \leq 0$ for efficiency, we put:

$$x_{\min} = 0 \text{ if } x_{\min} > 0$$

And recompute the optimal value for scale by setting $x_{\min}$ to 0, which yields:

$$\text{scale} = \frac{\sum_{i=1}^{n} l_i x_i}{\sum_{i=1}^{n} l_i^2} \text{ if } x_{\min} > 0$$

Nevertheless, the derived values for scale and min are not necessarily the most optimal. Since, we needed to start with initial values of scale and min to come up with more optimal ones, we may just have come up with a "local optimum". As this problem resembles a nested optimization problem, we can perform a grid search and search through different starting values of scale and min, optimize them and subsequently keep track of the reconstruction error. Finally, we can choose the values that yield the lowest MSE.

2 `void quantize_row_q4_K_reference`

The actual implementation to quantize a vector of weights is slightly more elaborate than the section above. Quantization in llama.cpp follows the concept of super-blocks and sub-blocks and can be summarized as follows:

- In the first step, a super block consisting of 256 weights is defined. This super block is divided into 8 sub-blocks, each containing 32 weights. For each sub-block i, the function `static float make_qkx2_quants` is applied and returns $\min_i$ and scale_i.
- The $\min_i$ and scale_i values are structured into the following vectors $\mathbf{S} = [s_1, s_2, s_3, \ldots, s_8]$ and $\mathbf{M} = [x_{min1}, x_{min2}, x_{min3}, \ldots, x_{min_8}]$ and quantized into k bits.
- The values scale_i and $\min_i$ are quantized using the scaling factors: $\frac{\max(\mathbf{S})}{2^k-1}$ and $\frac{\max(\mathbf{M})}{2^k-1}$ and clamped to $[0, 2^k - 1]$, which yields: $\mathbf{SQ} = [sq_1, sq_2, sq_3, \ldots, sq_8]$ and $\mathbf{MQ} = [xq_{min1}, xq_{min2}, xq_{min3}, \ldots, xq_{min8}]$.
- The dequantized values of $\mathbf{SQ}$ and $\mathbf{MQ}$ are subsequently used to quantize the initial weight vector $\mathbf{X} = [x_1, x_2, x_3, \ldots, x_n]$ leading to quantized weights $\mathbf{L} = [l_1, l_2, l_3, \ldots, l_n]$ stored using b bits.