GPTAQ

Neural network quantization with activations quantization and normalization techniques

Мета інформація

Тема моєї магістерської тези

Language model optimization using pruning, distillation and quantization techniques for NLP tasks

\/

GPTAQ - GPTQ + Activation quantization and cross layer equalization

Науковий керівник

Марченко Олександр Олександрович.

Постановка задачі

Оптимізувати метод запропонований в дослідженні GPTQ для квантизації (округлення/стиснення) нейронної мережі використовуючи, квантизацію активацій, та міжшарове вирівнювання для мінімізації впливу аутлаерів.

Стан проблеми і зміст роботи

Стан проблеми

- дослідження GPTQ опубліковано в березні 2023.
- дослідження Qualcom GPTVQ що базується на GPTQ опубліковане в лютому 2024 року - додає багатовимірність до квантизації, замінюючи округлення наближенням до центроїда що вираховується з врахуванням сусідніх значень.
- інші (https://arxiv.org/abs/2310.08041) - на зниженні ефекту аутлаерів при стисненні слоїв завдяки винесенню аутлаерів в інші слої

Короткий зміст роботи

- квантизації активацій, поряд із вагами, підвищує швидкість інференції завдяки нижчій точності для обчислень активацій.
- Міжшарове вирівнювання зменшує вплив аутлаерів при квантизації що зменшує вплив аутлаерів і покращує стабільність моделі.

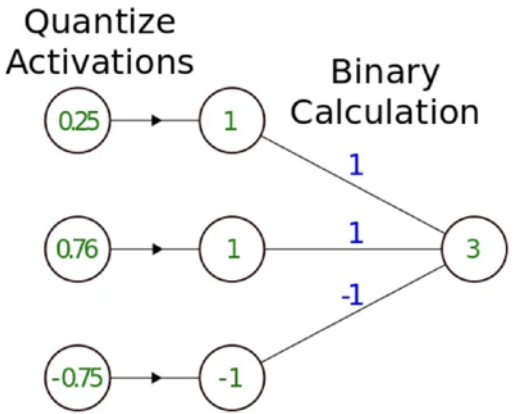
Quantization

0.91	0.74	0.03	0.54	0.99	0.18	0.39	0.58	0.69	0.57
0.47	0.19	0.55	0.92	0.46	0.93	0.03	0.61	0.65	0.97
0.94	0.04	0.76	0.40	0.35	0.13	0.61	0.51	0.09	0.33
0.36	0.58	0.58	0.50	0.36	0.19	0.51	0.70	0.13	0.43
0.15	0.67	0.95	0.34	0.78	0.14	0.15	0.16	0.16	0.17
0.29	0.72	0.14	0.39	0.65	0.89	0.37	0.65	0.69	0.89
0.28	0.93	0.24	0.08	0.04	0.33	0.68	0.18	0.53	0.01
0.89	0.93	0.99	0.73	0.11	0.53	0.85	0.96	0.89	0.24
0.18	0.32	0.49	0.27	0.60	0.52	0.21	0.22	0.69	0.73
0.82	0.12	0.97	0.13	0.89	0.95	0.46	0.24	0.70	0.50

➡️

46	37	2	27	50	9	19	29	34	28
24	10	28	46	23	47	2	31	33	48
47	2	38	20	17	7	31	26	5	16
18	29	29	25	18	10	25	35	6	22
7	34	48	17	39	7	7	8	8	9
15	36	7	20	32	44	19	32	34	44
14	47	12	4	2	17	34	9	27	0
45	46	50	36	6	26	43	48	44	12
9	16	24	13	30	26	11	11	35	36
41	6	48	7	44	47	23	12	35	25

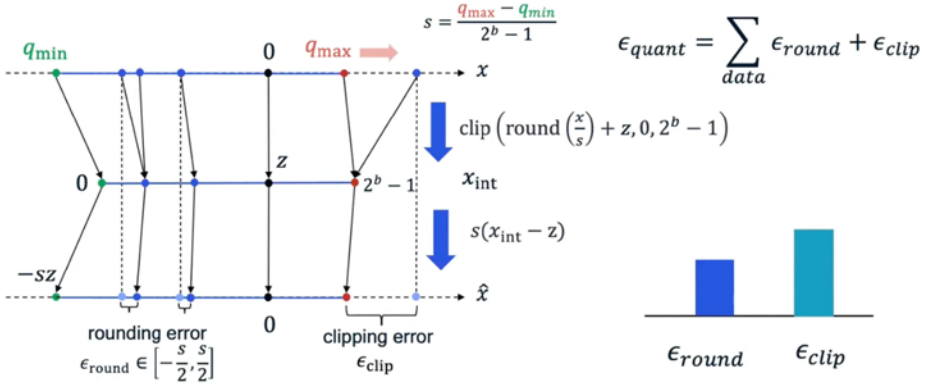
WEIGHTS

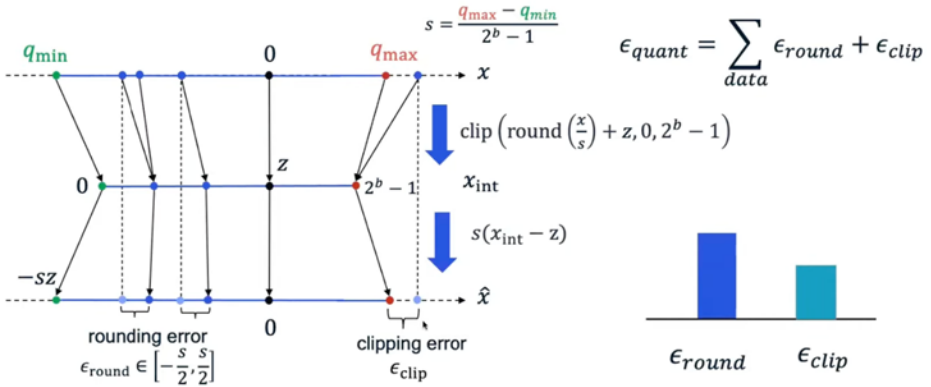


Quantization of activations, allowing binary calculations, with integer accumulation. The binary calculation can be convolutional layer or fully-connected layer.

Quantization

$$\text{clip}\left(\text{round}\left(\frac{x}{s}\right)+z, 0, 2^b-1\right)$$





GPTQ

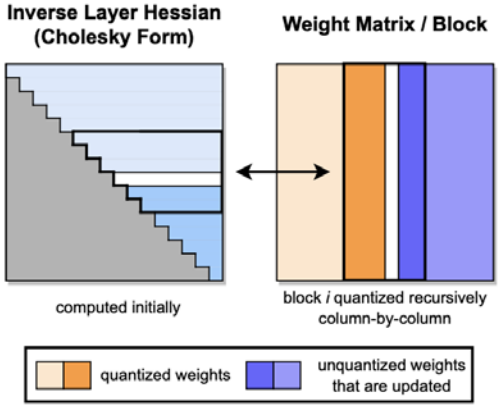


Figure 2: GPTQ quantization procedure. Blocks of consecutive *columns* (bolded) are quantized at a given step, using the inverse Hessian information stored in the Cholesky decomposition, and the remaining weights (blue) are updated at the end of the step. The quantization procedure is applied recursively inside each block: the white middle column is currently being quantized.

- OBQ (Hessian + round Q)
- +

1. Batch updates - columns in block B are processed, and are applied in one go to all columns outside of block B after all columns in block B are processed
2. Cholesky decomp. - Hessian (H^-1) - more numerically stable alternative Hessian row+col removal in OBQ

GPTQ

Algorithm 1 Quantize $\mathbf{W}$ given inverse Hessian $\mathbf{H}^{-1} = (2\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I})^{-1}$ and blocksize B.

$\mathbf{Q} \leftarrow \mathbf{0}_{d_{\text{row}} \times d_{\text{col}}}$ *// quantized output*
$\mathbf{E} \leftarrow \mathbf{0}_{d_{\text{row}} \times B}$ *// block quantization errors*
$\mathbf{H}^{-1} \leftarrow \text{Cholesky}(\mathbf{H}^{-1})^\top$ *// Hessian inverse information*
for $i = 0; B; 2B; \ldots$ **do**
  for $j = i; \ldots; i + B - 1$ **do**
    $\mathbf{Q}_{:,j} \leftarrow \text{quant}(\mathbf{W}_{:,j})$ *// quantize column*
    $\mathbf{E}_{:,j-i} \leftarrow (\mathbf{W}_{:,j} - \mathbf{Q}_{:,j}) / [\mathbf{H}^{-1}]_{jj}$ *// quantization error*
    $\mathbf{W}_{:,j:(i+B)} \leftarrow \mathbf{W}_{:,j:(i+B)} - \mathbf{E}_{:,j-i} \cdot \mathbf{H}^{-1}_{j,j:(i+B)}$ *// update weights in block*
  end for
  $\mathbf{W}_{:,(i+B):} \leftarrow \mathbf{W}_{:,(i+B):} - \mathbf{E} \cdot \mathbf{H}^{-1}_{i:(i+B),(i+B):}$ *// update all remaining weights*
end for

Hessian

$$\frac{\partial^2 f}{\partial x^2} = 2, \quad \frac{\partial^2 f}{\partial x \partial y} = 0, \quad \frac{\partial^2 f}{\partial x \partial z} = 0,$$
$$\frac{\partial^2 f}{\partial y^2} = 8, \quad \frac{\partial^2 f}{\partial y \partial x} = 0, \quad \frac{\partial^2 f}{\partial y \partial z} = 0,$$
$$\frac{\partial^2 f}{\partial z^2} = 2, \quad \frac{\partial^2 f}{\partial z \partial x} = 0, \quad \frac{\partial^2 f}{\partial z \partial y} = 0.$$

$$H = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

$$f(x, y, z) = x^2 + 4y^2 + z^2$$

Let $(x, y, z) = (1, 1, 1)$:

$$f(1, 1, 1) = 1^2 + 4(1^2) + 1^2 = 1 + 4 + 1 = 6$$

$$f(1, 2, 1) = 1^2 + 4(2^2) + 1^2 = 1 + 4(4) + 1 = 1 + 16 + 1 = 18$$

$$f(1, 0.5, 1) = 1^2 + 4(0.5^2) + 1^2 = 1 + 4(0.25) + 1 = 1 + 1 + 1 = 3$$

Why Hessian?

$\mathbf{Q} \leftarrow \mathbf{0}_{d_{\text{row}} \times d_{\text{col}}}$ // *quantized output*
$\mathbf{E} \leftarrow \mathbf{0}_{d_{\text{row}} \times B}$ // *block quantization errors*
$\mathbf{H}^{-1} \leftarrow \text{Cholesky}(\mathbf{H}^{-1})^{\top}$ // *Hessian inverse information*
for $i = 0; B; 2B; \ldots$ **do**
  for $j = i; \ldots; i + B - 1$ **do**
    $\mathbf{Q}_{:,j} \leftarrow \text{quant}(\mathbf{W}_{:,j})$ // *quantize column*
    $\mathbf{E}_{:,j-i} \leftarrow (\mathbf{W}_{:,j} - \mathbf{Q}_{:,j}) / [\mathbf{H}^{-1}]_{jj}$ ← 1 // *quantization error*
    $\mathbf{W}_{:,j:(i+B)} \leftarrow \mathbf{W}_{:,j:(i+B)} - \mathbf{E}_{:,j-i} \cdot \mathbf{H}^{-1}_{j,j:(i+B)}$ ← 2 // *update weights in block*
  end for
  $\mathbf{W}_{:,(i+B):} \leftarrow \mathbf{W}_{:,(i+B):} - \mathbf{E} \cdot \mathbf{H}^{-1}_{i:(i+B),(i+B):}$ ← 3 // *update all remaining weights*
end for

Hessian = sensitivity

1. error - divide by H[d,d] to amplify effects of error proportional to sensitivity

1. W upd (Block) - update leftover weights proportional to sensitivity

1. W upd - update all remaining weights to right of the Block proportional to sensitivity

Hessian - practically simplified

the weight matrix. As we are always dealing with a fixed layer ℓ, we drop the subscript ℓ to simplify notation. The objective is then equivalent to $\sum_{i=1}^{d_{\text{row}}} ||\mathbf{W}_{\mathbf{i},:}\mathbf{X} - \widehat{\mathbf{W}}_{\mathbf{i},:}\mathbf{X}||_2^2$.

This way of writing the error makes it clear that removing a single weight $[\mathbf{W}]_{ij}$ only affects the error of the corresponding output row $\mathbf{Y}_{\mathbf{i},:} = \mathbf{W}_{\mathbf{i},:}\mathbf{X}$. Hence, there is no Hessian interaction between different rows and so it suffices to work only with the individual $d_{\text{col}} \times d_{\text{col}}$ Hessians corresponding to each of the d_{row} rows. Further, as the dense layer output $\mathbf{Y} = \mathbf{WX}$ is fixed, the objective for each row has standard least squares form and its Hessian is given by $\mathbf{H} = \mathbf{2XX}^\top$.

```
XXX = torch.Tensor([[1,2,3], [4,5,6], [7,8,9]])
XXX @ XXX.T
```

✓ 0.0s

```
tensor([[ 14.,  32.,  50.],
        [ 32.,  77., 122.],
        [ 50., 122., 194.]])
```

Hessian -> Cholesky

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{00} & 0 & 0 \\ L_{10} & L_{11} & 0 \\ L_{20} & L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{00} & L_{10} & L_{20} \\ 0 & L_{11} & L_{21} \\ 0 & 0 & L_{22} \end{bmatrix}$$

GPTQ uses a Cholesky decomposition of the inverse Hessian H−1, which introduces a more numerically stable alternative to the inverse Hessian row and column removal operations of OBQ

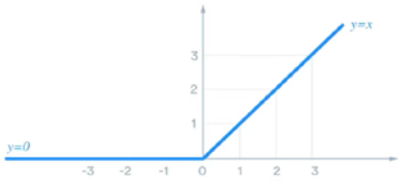
My work (1) - activations quant & CLE +bias absorption

6 Summary and Limitations

We have presented GPTQ, an approximate second-order method for quantizing truly large language models. GPTQ can accurately compress some of the largest publicly-available models down to 3 and 4 bits, which leads to significant usability improvements, and to end-to-end speedups, at low accuracy loss. We hope that our method will make these models accessible to more researchers and practitioners. At the same time, we emphasize some significant limitations: On the technical side, our method obtains speedups from reduced memory movement, and does not lead to computational reductions. In addition, our study focuses on generative tasks, and does not consider activation quantization. These are natural directions for future work, and we believe this can be achieved with carefully-designed GPU kernels and existing techniques (Yao et al., 2022; Wu et al., 2022).

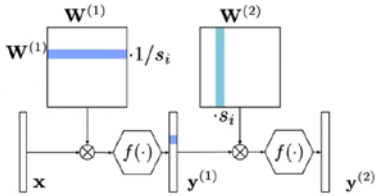
My work (1) - activations quant & CLE +bias absorption

Cross-layer equalization scales weights in neighbouring layers for better quantization



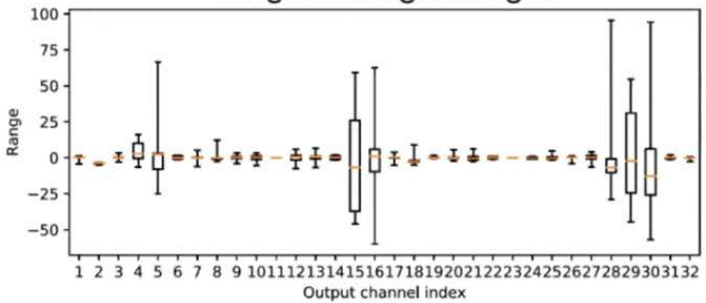
$\text{ReLU}(x) = \max(0, x)$

ReLU is scale-equivariant

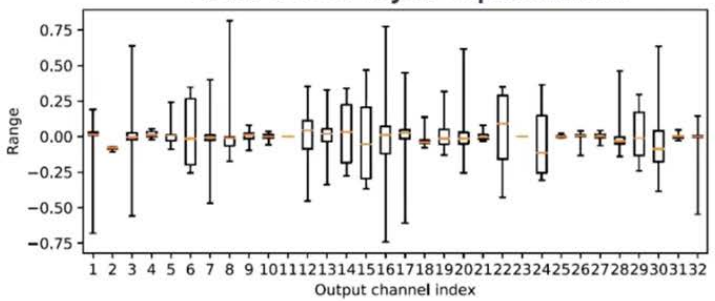


We can scale two neighboring layers together to optimize it for quantization

Original weight ranges



After cross-layer equalization



My work (2) - quantization granularity from Eigenvalues

Layer-Wise Quantization. At a high level, our method follows the structure of state-of-the-art post-training quantization methods (Nagel et al., 2020; Wang et al., 2020; Hubara et al., 2021; Frantar et al., 2022), by performing quantization layer-by-layer, solving a corresponding reconstruction problem for each layer. Concretely, let $\mathbf{W}_\ell$ be the weights corresponding to a linear layer ℓ and let $\mathbf{X}_\ell$ denote the layer input corresponding to a small set of m data points running through the network. Then, the objective is to find a matrix of quantized weights $\widehat{\mathbf{W}}$ which minimizes the squared error, relative to the full precision layer output. Formally, this can be restated as

$$\text{argmin}_{\widehat{\mathbf{W}}} ||\mathbf{W}\mathbf{X} - \widehat{\mathbf{W}}\mathbf{X}||_2^2. \quad (1)$$

- Optimization-based methods:

$$\text{argmin}_{q_{\min}, q_{\max}} \ell\left(\boldsymbol{X}, \widehat{\boldsymbol{X}}(q_{\min}, q_{\max})\right)$$

MSE Cross-entropy

Right now quantiazation min-max range is set from MSE in the step before quantization

Idea is to use eigenvalues from diagonal Hessian to improve quant scale depending on sensitivity of weight (eigenvalue of Hessian)

```
eigenvalues = eigenvalues_from_H(H)
self.scale = adjust_scale(self.scale, eigenvalues)
```

EXPERIMENTS

Method	Inference Speed (s)	Percentage of Baseline (%)
W4_EIG_CUDA	0.011	100.0
W4_A4_RTN	0.012	100.924
W4_A4_TOKEN	0.011	98.239
W4_A4_REOPTIMIZE	0.011	100.656
W4_A4_CLE	0.011	100.675
W4_A4_EIG	0.017	151.416
W4_A4_RTN_CLE	0.012	101.014
W4_A4_RTN_EIG	0.012	102.261

Model	Relative Increase (%)
Original (Baseline)	0
W4	17.04
W4_A4_RTN	6.82
W4_A4_Token	6.84
W4_A4_Reoptimize	6.85
W4_A4_EIG	53.23
W4_A4_RTN_CLE	1.95
W4_A4_RTN_EIG	53.22

ABLATION

RTN with CLE

Combining RTN activation quantization with weight quantization.

Model	Relative Increase (%)
W4_A4_RTN_CLE	1.95

RTN with Eigenvalues

Combining RTN activation quantization with eigenvalue-based quantization.

Model	Relative Increase (%)
W4_A4_RTN_EIG	53.22

Thanks for attention