GPTQ

December 16, 2024

1 Abstract

Optimal brain quantization is an approach to optimizing neural network size by removing unimportant weights. It aims to compress the model as much as possible, while keeping the increase in error small. The Optimal Brain Surgeon (OBS) method calculates the inverse Hessian matrix from training data and structural information of the net to prune more weights than other methods for better generalization on test data. In Optimal Brain Compression the authors introduced a new compression framework that operates independently layer-by-layer and covers both weight pruning and quantization in a unified setting, suitable for modern deep networks. GPTQ further optimizes this approach by changing the order of operations, such that the Hessian updates can be efficiently vectorized and parallelized, and employing GPU-specific tweaks, making OBQ suitable for large models with negligible accuracy degradation. These studies demonstrate the potential for optimal brain quantization to improve the efficiency and performance of neural networks across various applications. Our project follows the full evolution of the optimal quantization methods, from the underlying optimization theory to modern tweaks and improvements.

2 Problem background

2.1 Layer output approximation

Pruning is a procedure that aims to remove most of the weights in a neural network by setting them to zero, without sacrificing the performance too much. Moreover, in some cases pruning can lead to improved generalization on test data, acting as a regularization. Quantization decreases the model complexity and size by mapping all weights to a discrete representation, typically an integer data type of a small size. The advantage over pruning is the simplicity of storage (no need for sparse matrices) and improved inference speed.

The modern problem statement for optimal post training pruning and quantization is based on approximating the outputs of each individual layer in a network. A layer ℓ is modeled as a function $f_\ell(\mathbf{X}_\ell, \mathbf{W}_\ell)$ acting on inputs $\mathbf{X}_\ell$, parametrized by weights $\mathbf{W}_\ell$. For compressed weights $\widehat{\mathbf{W}}_\ell$, a loss $\mathcal{L}$ and a generic compression constraint $\mathcal{C}(\widehat{\mathbf{W}}_\ell) > C$, we try to minimize this objective

$$\text{argmin}_{\widehat{\mathbf{W}}_\ell} \ \mathbb{E}_{\mathbf{X}_\ell} \mathcal{L}(f_\ell(\mathbf{X}_\ell, \mathbf{W}_\ell), f_\ell(\mathbf{X}_\ell, \widehat{\mathbf{W}}_\ell)) \quad \text{s.t.} \quad \mathcal{C}(\widehat{\mathbf{W}}_\ell) > C$$

For $d_{\text{row}} \times d_{\text{col}}$ weights $\mathbf{W}_\ell$ and the input $\mathbf{X}_\ell$ has dimensions $d_{\text{col}} \times N$ we minimize [1] [3]

$$\text{argmin}_{\widehat{\mathbf{W}}_\ell} \ ||\mathbf{W}_\ell \mathbf{X}_\ell - \widehat{\mathbf{W}}_\ell \mathbf{X}_\ell||_2^2 \quad \text{s.t.} \quad \mathcal{C}(\widehat{\mathbf{W}}_\ell) > C. \tag{1}$$

The exact solution is an NP-hard problem.

2.2 Greedy approach in OBS and OBQ

Optimal Brain Damage (OBD) introduces an approach to pruning relying on a local Taylor expansion of the neural network error [1]. For computational simplicity, OBD assumes that the Hessian matrix is diagonal; in fact, however, neural network loss landscape is highly non-convex, therefore Hessians tend to be strongly non-diagonal . This leads OBD to eliminate the wrong weights [2].

Optimal Brain Surgeon (OBS) does not make any assumptions about Hessian, instead opting to recalculate the magnitude of the remaining weights when a single weight is removed. Similar to OBD, the authors start from Taylor expansion of the training error of a neural network [2]

$$\delta E = \left(\frac{\partial E}{\partial \mathbf{w}}\right) \delta \mathbf{w} + \frac{1}{2}\delta \mathbf{w}^T \mathbf{H} \delta \mathbf{w} + O\left(\|\delta \mathbf{w}\|^3\right) \tag{2}$$

where $\mathbf{H}$ is the Hessian matrix. For a network trained to a local minimum, the first order term vanishes. The goal is to set one of the weights (w_p) to zero by adding a small $\delta \mathbf{w}$. A constrained optimization problem arises:

$$\min_p \min_{\delta \mathbf{w}} \frac{1}{2}\delta \mathbf{w}^T \mathbf{H} \delta \mathbf{w}, \text{ such that } \mathbf{e}_p^T \delta \mathbf{w} + w_p = 0 \tag{3}$$

We can write the Lagrangian for this problem

$$L(\delta \mathbf{w}, \lambda) = \frac{1}{2}\delta \mathbf{w}^T \mathbf{H} \delta \mathbf{w} + \lambda(\mathbf{e}_p^T \delta \mathbf{w} + w_p) \tag{4}$$

Taking functional derivatives yields (not in the paper, done manually)

$$\nabla_{\delta \mathbf{w}} L = \mathbf{H}\delta \mathbf{w} + \lambda \mathbf{e}_p \tag{5}$$

$$\nabla_{\lambda} L = \mathbf{e}_p^T \delta \mathbf{w} + w_p \tag{6}$$

Solving for an extremum, employing the constraints of 3, and using matrix inversion one can find that the optimal weight change and resulting change in error are

$$\begin{aligned} \delta \mathbf{w} &= -\lambda \mathbf{H}^{-1} \mathbf{e}_p \\ \delta \mathbf{w}_p &= -w_p \;\Rightarrow \\ \lambda &= w_p / [\mathbf{H}^{-1}]_{pp} \end{aligned}$$

$$\delta \mathbf{w} = -\frac{w_p}{[\mathbf{H}^{-1}]_{pp}} \cdot \mathbf{H}^{-1} \mathbf{e}_p, \quad w_p = \text{argmin}_{w_p} \frac{w_p^2}{[\mathbf{H}^{-1}]_{pp}}$$

However, in this fashion only one weight can be pruned optimally. A greedy algorithm is employed to successfully remove a significant portion of the weights. An optimal index p that gives a minimal increase in error is chosen at each step, and optimal $\delta \mathbf{w}$ is computed. Nevertheless, this does not provide any guarantee of the global optimality [2].

Optimal Brain Compression (OBC) introduces a unified framework both for pruning (OBS) and quantization (OBQ). Here, the outputs of individual layers are approximated instead of the full training loss, which enables computation of Hessians for each layer separately instead of computing the full Hessian for very large models. The outputs are compared with $\mathcal{L}_2$ norm. Likewise, for weight quantization we can write

$$L(\delta \mathbf{w}, \lambda) = \frac{1}{2}\delta \mathbf{w}^T \mathbf{H} \delta \mathbf{w} + \lambda(\mathbf{e}_p^T \delta \mathbf{w} + w_p - \text{quant}(w_p)) \tag{7}$$

so that $\delta \mathbf{w}_p + w_p = \text{quant}(w_p)$ the small addition quantizes the weight instead of zeroing it. The solution is analogous

$$\delta \mathbf{w} = -\frac{w_p - \text{quant}(w_p)}{[\mathbf{H}^{-1}]_{pp}} \cdot \mathbf{H}^{-1} \mathbf{e}_p, \quad w_p = \text{argmin}_{w_p} \frac{(w_p - \text{quant}(w_p))^2}{[\mathbf{H}^{-1}]_{pp}} \tag{8}$$

The obvious challenge in applying the OBS framework in its true form, i.e. pruning a single weight at a time using the exact formulas in (8), is computationally very demanding. The Hessian $\mathbf{H}$ is a $d \times d$, which is already expensive to store and compute with. Additionally, this matrix needs to be updated and inverted at each of the $O(d)$ steps with a computational complexity of $O(d^3)$.

The authors rewrite the $\mathcal{L}_2$ objective as a sum of the squared errors for each row in the weight matrix. They notice that each row of the weight matrix can be quantized in parallel, sharing Hessian updates between rows. The greedy algorithm can therefore be applied to each row independently. Moreover, inverse Hessian updates can now be computed efficiently by simple Gaussian elimination, which is formulated in the following lemma.

Lemma 2.1. *Given an invertible $d_{col} \times d_{col}$ matrix $\mathbf{H}$ and its inverse $\mathbf{H}^{-1}$, we want to efficiently compute the inverse of $\mathbf{H}$ with row and column p removed, which we denote by $\mathbf{H}_{-p}$. This can be accomplished through the following formula:*

$$\mathbf{H}_{-p}^{-1} = \left(\mathbf{H}^{-1} - \frac{1}{[\mathbf{H}^{-1}]_{pp}} \mathbf{H}^{-1}_{:,p} \mathbf{H}^{-1}_{p,:}\right)_{-p}, \tag{9}$$

which corresponds to performing Gaussian elimination of row and column p in $\mathbf{H}^{-1}$ followed by dropping them completely. This has $\Theta(d_{col}^2)$ time complexity.

Algorithm 1 Quantize $k \leq d_{\text{col}}$ 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**
  $p \leftarrow \text{argmin}_{p \in M} \frac{1}{[\mathbf{H}^{-1}]_{pp}} \cdot (\text{quant}(w_p) - w_p)^2$
  $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}^{-1}_{:,p} \frac{1}{[\mathbf{H}^{-1}]_{pp}} \cdot (\text{quant}(w_p) - w_p)$
  $\mathbf{H}^{-1} \leftarrow \mathbf{H}^{-1} - \frac{1}{[\mathbf{H}^{-1}]_{pp}} \mathbf{H}^{-1}_{:,p} \mathbf{H}^{-1}_{p,:}$
  $M \leftarrow M - \{p\}$
end for

2.3 GPTQ

GPTQ is based on OBQ, therefore the math of GPTQ is the same as OBQ. However, GPTQ features 3 major engineering changes that make possible quantization of very large models efficiently utilizing GPU.

1. Fixed quantization order
2. Blockwise update and correction accumulation
3. Cholesky reformulation of Hessian update

GPTQ authors argue that, particularly for quantization, the order of quantization determined by a greedy algorithm is not critical. They find that its improvement over quantizing the weights in arbitrary order is generally small, in particular on large, heavily-parametrized layers. Most likely, this is because the slightly lower number of quantized weights with large individual error is balanced out by those weights being quantized towards the end of the process, when only few other unquantized weights can be adjusted [4].

GPTQ opts to quantize all rows in the same order. This results in $O(d_{row})$ Hessian updates, thus the final GPTQ time complexity is $O(\max(d_{row}d_{col}^2, d_{col}^3))$, provide a one magnitude speed up.

Next, note that update of all the unquantized columns is an operation that does a small number of FLOPS, yet heavily uses memory access. Thus, to reduce memory throughput GPTQ suggests to split columns in blocks of size B, run the fixed order quantization with update of columns inside the current block, while accumulating the correction. After the block is fully quantized, the correction is applied to the rest of the columns.

The third improvement is related to the inverse Hessian update in Lemma 2.1. The operations in this computation is ill conditioned. For large models, the numerical error always explodes the Hessian update. GPTQ uses more computationally stable algorithm, namely Cholesky decomposition, to precomute rows of the inverse Hessian update. Indeed, the only information required from $\mathbf{H}_{F_q}^{-1}$, where F_q denotes the set of unquantized weights when quantizing weight q, is the elements in q-th row starting with the diagonal. Then, the row removal for symmetric $\mathbf{H}^{-1}$ corresponds to taking a Cholesky decomposition, except for the minor difference that the latter divides row q by $\sqrt{[\mathbf{H}_{F_q}^{-1}]_{qq}}$.

Algorithm 2 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}
$\mathbf{H}^{-1} \leftarrow \text{Cholesky}(\mathbf{H}^{-1})$ {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}
    $\mathbf{W}_{:,j:(i+B)} \leftarrow \mathbf{E}_{:,j-i} \cdot \mathbf{H}^{-1}_{j,j:(i+B)}$ {update weights in block}
  end for
  $\mathbf{W}_{:,(i+B):} \leftarrow \mathbf{E} \cdot \mathbf{H}^{-1}_{i:(i+B),(i+B):}$ {update all remaining weights}
end for

3 Experiments

As an example we took LLaMA-2 7B. This LLM is fairly large, yet it fits our kaggle P100 16GB GPU. Then we compared perplexity of round-to-nearset (RTN) and GTPQ quantizations of the model. Also, for the reference we measured perplexity for half-precision model (FP16).

The perplexity is often used to evaluate the performance of LLM and is defined as

$$\text{Perplexity} = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 P(w_i|w_1,w_2,\ldots,w_{i-1})}$$

Where N is the total number of words in the dataset, $P(w_i \mid w_1, w_2, \ldots, w_{i-1})$ is the probability of the i-th word given the preceding context, as predicted by the language model. In general, the lower the perplexity the better the model.

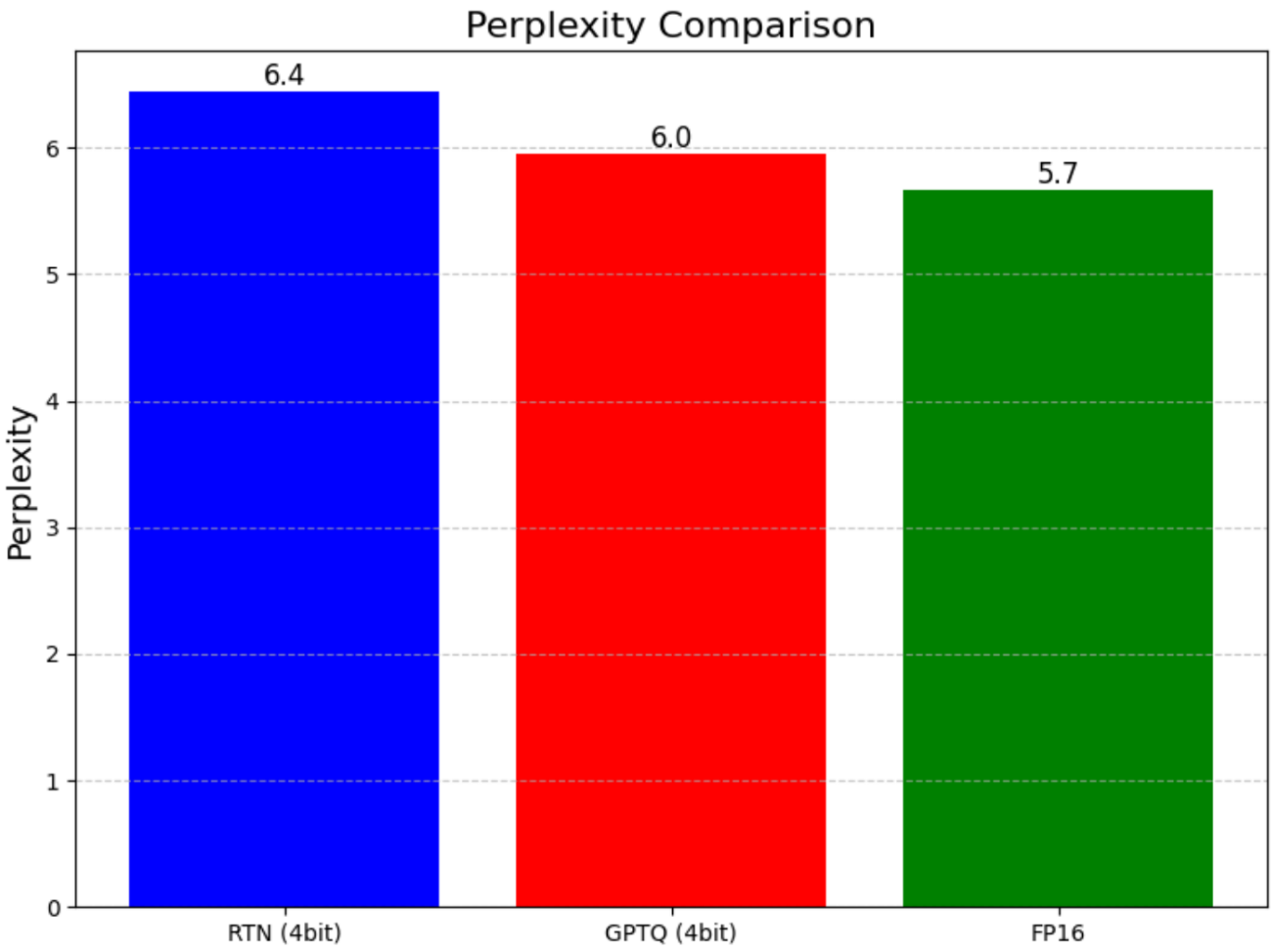

Figure 1: Perplexity of 4 bit quantizations RTN, GPTQ, and half-precision models.

Due to a magnitude speed up of GPTQ, the quantization of 7B model was finished in 30 minutes on a simple P100 GPU. GPTQ outperforms RTN in terms of perplexity and saves 4 times memory compared to half-precision FP16 model.

4 Contribution

Foma: abstract, problem background; OBS research and the related part of the presentation.
Kamil: code and experiments; GPTQ problem background; problem statement, OBQ, GPTQ research and the related part of the presentation.

References

[1] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2, Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

[2] B. Hassibi and D. Stork, "Second order derivatives for network pruning," 1993.

[3] E. Frantar, S. Pal Singh, and D. Alistarh, "Optimal brain compression: A framework for accurate post-training quantization and pruning," 2021.

[4] S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," 2022.