Model-Preserving Adaptive Rounding

Albert Tseng
Zhaofeng Sun
Chris De Sa

We present Yet Another Quantization Algorithm (YAQA), which gives the 1st E2E quantization error bounds and achieves SOTA quality.

LLM Post Training Quantization

The goal of LLM quantization is to preserve the output of the original unquantized model:

$$\hat{\theta} \leftarrow \arg\min_{\theta^* \in C} \mathbb{E}_{X\sim\mathscr{D}}\left[D_{KL}(M(\theta^*, X)\|M(\theta, X))\right]$$

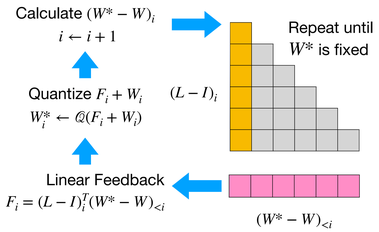
Unfortunately, solving this problem is NP-hard. One common approximation algorithm is to instead perform layerwise adaptive rounding, which uses information from the Hessian of the KL to round each layer $W \in \mathbb{R}^{m\times n}$ independently:

$$\mathbb{E}_{x\sim\mathscr{D}} D_{KL}\left(M(W^*, x)\,||\,M(W, x)\right) \approx (W^* - W)\nabla\ell(W) + \frac{1}{2}(W^* - W)\nabla^2\ell(W)(W^* - W)^T$$
$$= \frac{1}{2}(W^* - W)\nabla^2\ell(W)(W^* - W)^T$$

State of the art PTQ algorithms like LDLQ from QuIP/QuIP#/QTIP (and equivalently GPTQ) fall under this framework, using linear feedback from a structured approximation H of $\nabla^2\ell(W)$ to quantize. LDLQ runs the following iterative rounding procedure using the Cholesky of H:

$$\nabla^2\ell(W) \approx H \in \mathbb{R}^{mn\times mn}$$
$$H \leftarrow LDL^T$$
$$W^* = \text{vec}^{-1}\left(\mathbb{Q}\left(\text{vec}(W) + \text{vec}(W^* - W)(L - I)\right)\right)$$

This iteratively rounds W_i using linear feedback from $W_{<i}$: $W_i^* = \mathbb{Q}(W_i + (L-I)_i^T(W^* - W)_{<i})$

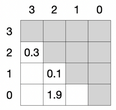


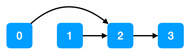
Since the quantized model W^* is fully determined by H, this raises 2 questions:

1. What structured approximations H of $\nabla^2\ell(W)$ make ★ tractable?
2. How well does adaptive rounding with H minimize $\mathbb{E}_{X\sim\mathscr{D}}\left[D_{KL}(M(W^*, X)\|M(W, X))\right]$?

Near-Optimal Hessian Sketches

To answer Q1, we show that the support of $L - I$ forms a weight dependency graph for adaptive rounding; adaptive rounding runs in the depth of this graph. We formalize this with the SND, which tells us how many serial steps adaptive rounding will converge in for a given H.





Definition: **(Structural Nilpotence Degree (SND))** *Let L be a unit triangular matrix. $\text{snd}(L)$ is the degree of the binary nilpotent matrix N with the same support (nonzero mask) as $L - I$, where the degree of N is the smallest k such that $N^k = 0$.*

Lemma: *Adaptive rounding converges in at most $\text{snd}(L)$ steps*

$\nabla^2\ell(W)$ is empirically ≈ low rank, so a natural choice is a Kronecker-factored $H \approx H_O \otimes H_I$, which has SND $= m + n$. In contrast, GPTQ's $H = I \otimes \mathbb{E}[xx^T]$ has a SND $= n$, which is cheaper but, as we show later, results in a significantly worse H and quantized model! Using $H = H_O \otimes H_I$ lets us bound the second order approximation of the KL, which answers Q2:

Theorem 1. *Let $H_O \in \mathbb{R}^{m\times m}$ and $H_I \in \mathbb{R}^{n\times n}$ be two p.d. matrices, Q perform nearest or stochastic rounding with $\mathbb{E}[(Q(x) - x)^2] \le \sigma^2$, and $\Delta = W^* - W$. Then,*

$$\text{vec}(\Delta)\nabla^2\ell(W)\text{vec}(\Delta)^T \le \|\nabla^2\ell(W)\|_F\left(\|\Delta\|_F^2\sqrt{2-2c} + \mu_I^2\mu_O^2\frac{\text{tr}(H_I^{1/2})^2\text{tr}(H_O^{1/2})^2}{mn\|H_I\|_F\|H_O\|_F}\sigma^2\right)$$

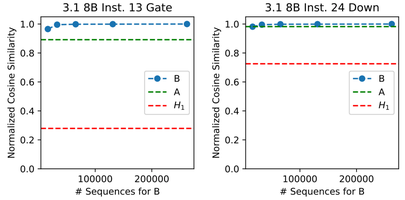
where $c = \dfrac{\langle\nabla^2\ell(W), H_O\otimes H_I\rangle}{\|\nabla^2\ell(W)\|_F\|H_O\|_F\|H_I\|_F}$ is the cosine similarity between $\nabla^2\ell(W)$ and $H_O\otimes H_I$.

Theorem 1 shows that we should maximize c, which we can do with power iteration. The Hessian of the KL is the FIM $\nabla^2\ell(W) = \mathbb{E}\left[\text{vec}(\nabla c(W))\text{vec}(\nabla c(W))^T\right]$, which gives two sketches:

Sketch A assumes tokens in a sequence are independent (not true for LLMs!) and power iterates on $\mathbb{E}[xx^T \otimes (\nabla_y c)(\nabla_y c)^T]$. This reduces variance at the cost of bias.

Sketch B performs one round of power iteration with an identity initialization, which gives $(H_O)_1 \leftarrow \mathbb{E}[(\nabla c)(\nabla c)^T]$, $(H_I)_1 \leftarrow \mathbb{E}[(\nabla c)^T(\nabla c)]$. This can be computed in a single pass!

Both are designed to be easily computable with modern distributed training frameworks like PyTorch FSDP, and importantly, both have much higher cosine similarity than LDLQ and GPTQ's Hessian sketch $H_1 = I \otimes \mathbb{E}[xx^T]$!



SOTA Quantization Quality

YAQA's Hessian sketches achieve much higher cosine similarity to $\nabla^2\ell(W)$ than LDLQ's Hessian, which directly translates to state of the art quantization quality and a 30% reduction in KL. YAQA also outperforms GuidedQuant, which can also be expressed in the SND framework! (QTIP + IP, no FT)

Algo.	Bits	D_{KL} ↓	PPL ↓		0-shot ↑	D_{KL} ↓	PPL ↓		0-shot ↑
		W2	W2	C4	Avg	W2	W2	C4	Avg
		3.1 70B Inst.				3.2 3B Inst.			
BF16	16	0	3.52	6.46	67.67	0	9.58	10.62	63.79
LDLQ	2	0.497	6.02	7.82	65.45	0.455	15.30	14.69	55.68
	3	0.138	4.26	6.74	67.58	0.085	10.69	11.44	62.46
	4	0.045	3.74	6.54	67.58	0.021	**9.78**	10.79	63.70
GuidedQuant	2	0.383	5.59	7.51	65.44	0.342	13.56	13.96	57.72
	3	0.111	4.10	6.68	67.33	0.070	10.48	11.19	63.48
	4	0.035	3.72	6.53	**67.80**	0.019	9.84	10.80	**64.10**
YAQA-A	2	0.378	5.56	7.51	65.92	0.333	13.75	13.56	58.29
	3	0.110	4.10	6.68	**67.69**	0.059	10.37	11.14	62.51
	4	0.036	3.73	6.52	67.50	0.015	**9.78**	10.76	63.28
YAQA-B	2	**0.335**	**5.30**	**7.34**	**66.19**	**0.288**	**13.18**	**13.10**	**59.11**
	3	**0.094**	**4.01**	**6.64**	67.24	**0.047**	**10.15**	**11.09**	**62.74**
	4	**0.030**	**3.69**	**6.51**	67.73	**0.014**	9.80	**10.75**	63.31

Even with recovery finetuning, YAQA significantly improves performance over LDLQ (QTIP + IP + FT)

Algo.	Bits	D_{KL} ↓	PPL ↓		0-shot ↑	D_{KL} ↓	PPL ↓		0-shot ↑
		W2	W2	C4	Avg	W2	W2	C4	Avg
		Llama 3.1 70B Instruct				Llama 3.1 8B Instruct			
BF16	16	0	3.52	6.46	67.67	0	6.50	8.02	69.82
LDLQ	2	0.302	5.01	7.16	66.11	0.185	7.82	9.20	65.44
	3	0.101	3.96	6.64	**67.46**	0.048	6.80	8.31	68.42
	4	0.036	3.71	6.54	67.64	0.016	6.61	8.13	69.47
YAQA-A	2	0.279	4.92	7.10	66.26	0.163	7.63	9.06	**67.54**
	3	0.098	3.88	6.63	66.94	0.042	6.78	8.28	**69.23**
	4	0.032	3.68	**6.52**	67.59	0.014	6.58	**8.10**	69.50
YAQA-B	2	**0.266**	**4.82**	**7.07**	**66.99**	**0.147**	**7.60**	**8.96**	66.38
	3	**0.091**	**3.87**	**6.61**	67.42	**0.038**	**6.74**	**8.27**	68.88
	4	**0.029**	**3.67**	**6.52**	**67.69**	**0.012**	**6.56**	8.11	**70.12**

YAQA even achieves a lower KL than Google's QAT! (Gemma 3 12B Inst., INT4 + IP, no FT)

Algo.	Quant. Type	Bits	D_{KL} ↓	PPL ↓		0-shot Acc ↑					
			W2	W2	C4	Avg	ArcC	ArcE	BoolQ	HSwag	PIQA
BF16	None	16	0	7.85	8.61	70.22	54.01	78.79	87.25	54.27	76.77
Google QAT	QAT	4.5	0.089	7.56	8.52	70.83	54.52	79.76	86.82	54.77	78.29
YAQA-A	PTQ	4	0.058	7.96	8.69	70.12	53.90	78.83	87.09	53.68	77.09
YAQA-B	PTQ	4	0.056	7.94	8.67	69.90	54.10	78.66	86.91	54.13	75.68

Although YAQA uses gradients, using as few as 2K sequences gives significantly better results than LDLQ while also being cheaper! (Qwen 3 8B Inst., QTIP + IP, no FT)

Qwen3 8B	Seqs.	GPU-Hrs	Bits	W2 PPL (4K ctx)	W2 KL (4K ctx)	MMLU 0 shot	GSM8K 4 shot CoT Exact Match
Original			16	9.00	0	72.99	84.31
LDLQ	4K	1.5	2	10.81	0.285	63.94	56.31
YAQA-B	2K	1	2	10.33	0.227	65.74	62.14
YAQA-B	4K	2	2	10.32	0.221	66.04	63.82
YAQA-B	8K	4	2	10.34	0.212	66.65	63.34
YAQA-B	16K	7	2	10.15	0.210	66.29	64.86
YAQA-B	32K	15	2	10.20	0.206	66.59	67.02
YAQA-B	64K	30	2	10.16	0.205	66.75	66.07