BoA-Based Post-Training Quantization of OPT-125M: A Practical Implementation Study and Deviations from the Official Algorithm

Amitesh Shekhar, Toshan Achintya Golla, Vatsal Melwani
Private project, Kaggle environment (NVIDIA P100 GPU)

***Abstract*—We re-implement the BoA (Attention-aware Post-training Quantization without Backpropagation) idea and apply it to OPT-125M on Kaggle (NVIDIA P100). Our goal is to reproduce BoA's improvements over GPTQ and to evaluate practical engineering trade-offs necessary to run the algorithm under limited GPU memory. We describe the core algorithmic ideas implemented, list the differences from the official BoA code and paper, and present quantitative results (PPL, ARC, SQuAD-F1, BLEU, ROUGE-L) for FP16 and quantized models (INT4, INT2). We analyze why our results differ from the numbers reported in the original BoA paper and provide a roadmap of fixes and improvements required to fully match the published performance.**

***Index Terms*—post-training quantization, BoA, GPTQ, OPT-125M, low-bit quantization, language models**

I. Introduction

Large language models (LLMs) provide state-of-the-art performance across many NLP tasks but are increasingly expensive to store and run. Post-training quantization (PTQ) is an attractive avenue for reducing model size and inference cost without expensive retraining. GPTQ [2] demonstrated accurate one-shot quantization for generative transformers by using approximate second-order (Hessian) information and a sequential column-wise quantization procedure. More recently, the BoA algorithm proposed attention-aware Hessians to capture inter-layer dependencies in the attention module and significantly improve accuracy at low bit-widths [1].

We re-implemented a practical variant of BoA and applied it to OPT-125M [3]. Our implementation was adapted to run on Kaggle (single NVIDIA P100 GPU), which necessitated multiple engineering trade-offs. We present our implementation, report results for INT4 and INT2 quantization, and compare against FP16 baselines.

II. Background

A. GPTQ recap

GPTQ [2] approximates the layer-wise reconstruction loss

$$\|Q(W)X - WX\|_F^2,$$

and builds a column Hessian $H_{\text{col}} \approx X^\top X$. GPTQ uses a Cholesky-based inversion of H_{col} and sequentially quantizes columns, updating remaining columns using a Hessian-informed correction to account for quantization error.

B. BoA (high level)

BoA [1] extends GPTQ by introducing attention-aware Hessians that consider inter-layer dependencies within the attention module. The BoA Hessian is approximated as a Kronecker-structured object:

$$H \approx H_{\text{col}} \otimes H_{\text{row}},$$

where H_{row} encodes attention-specific dependencies derived from Q/K/V and attention activations. This allows head-wise joint updates along rows (per-head) in addition to the GPTQ column updates.

III. Implementation

A. Code base and environment

We implemented the algorithm in PyTorch and used the HuggingFace Transformers library to load OPT-125M [3], [4]. The notebook ran on Kaggle with a single NVIDIA P100 GPU; memory constraints shaped many engineering choices described below.

B. Calibration and evaluation protocol

Calibration was performed on WikiText-2 using the same token sampling strategy as the paper: NSAMPLES=128, SEQ_LEN=2048. Evaluation metrics: perplexity on WikiText, ARC (Easy / Challenge), SQuAD-F1, and CNN/DailyMail BLEU-1 and ROUGE-L.

C. Algorithmic implementation (what we implemented)

Our quantizer contains two interacting components:

- **Column Hessian (GPTQ part):** we accumulate $H_{\text{col}} = \sum X^\top X$ over calibration batches for each layer, perform a Cholesky factorization, compute an inverse approximation H_{col}^{-1}, and use the standard GPTQ sequential column quantization update.
- **Row Hessian (BoA part):** for Q and K projections, we additionally accumulate per-head row Hessians (e.g., $Q_h^\top Q_h$ or $K_h^\top K_h$); we compute per-head Cholesky and derive per-head factors $U_{\text{row},h}$ used to perform a head-wise row update during quantization (see Algorithm 1 in [1]).

D. Engineering and memory optimizations

To run on a P100 with limited memory we used:

- **Layer-by-layer GPU loading**: move a single layer to GPU for calibration/quantization and move it back to CPU after quantization.
- **Capture-first-layer activations**: run a single pass to capture inputs to the first layer, then forward layer-by-layer (recomputing activations only for one layer at a time).
- **Float32 Hessians on CPU**: keep Hessians and Cholesky computations on CPU (float32) to reduce GPU memory footprint.
- **Aggressive cleanup**: use `gc.collect()` and `torch.cuda.empty_cache()` between layers.

E. Key deviations from the official BoA implementation

For reproducibility and clarity these differences must be explicitly stated in the paper:

1) **Quantization grid (qparams):** official BoA uses an MSE/Hessian-weighted grid search for scale/zero ("Hessian" qparam_comput); our implementation used a per-group MinMax approach for scale/zero (fast but suboptimal at low bits).
2) **Bits wiring bug:** originally the code used a default bits value in some internal calls; ensure your run sets the per-quantizer bits parameter correctly (we corrected this later). Earlier runs reported as ÏNT3ẅere actually quantized with 4-bit grids.
3) **Partial BoA usage:** we applied BoA-style row Hessians for Q/K projections only; V and out projections were quantized with GPTQ fallback.
4) **Row update approximation:** our row update uses per-head U_{row} factors but does not multiply with an explicit U_{col} term as in the full Kronecker-aware update, producing a practical but approximate correction.
5) **Ordering heuristics not used:** we did not run act_order_row / act_order_col variants and did not select the best-of-two runs as the reference implementation sometimes does.
6) **Hard-coded head count:** implementation assumed 12 attention heads; parameterize this for general use.

IV. Experiments

A. Setup

Model: OPT-125M [3]. Calibration: WikiText-2 (NSAMPLES=128, SEQ_LEN=2048). Hardware: Kaggle single NVIDIA P100 GPU.

B. Results

Table I summarizes the main evaluation metrics you provided for FP16 and quantized models (INT4 and INT2). These are reproduced directly from the runs.

TABLE I: Evaluation summary for FP16 baseline vs. BoA-quantized OPT-125M. Metrics: WikiText Perplexity (PPL), ARC accuracies, SQuAD-F1, and CNN/DailyMail BLEU-1/ROUGE-L.

Model	PPL	ARC-E	ARC-C	SQuAD-F1	BLEU	R-L
FP16	27.62	0.425	0.260	0.000	0.0262	0.0987
INT4	30.41	0.440	0.225	0.000	0.0259	0.0989
INT3	51.38	0.385	0.270	0.000	0.0260	0.0958
INT2	7092.32	0.290	0.245	0.000	0.0330	0.1204

C. Short analysis of results

- INT4 shows a modest increase in PPL (+10%) while keeping ARC and summarization metrics near baseline; this suggests the quantizer preserved many predictive properties of the model at 4 bits.
- INT2 exhibits catastrophic PPL degradation ($27 \rightarrow 7092$), while some task-level metrics (BLEU) show noisy behavior — probably reflecting degenerate generation rather than preserved language understanding.
- The observed differences from the published BoA numbers (e.g., INT3 PPL reported lower in the paper) are explained by our implementation differences: MinMax qparams rather than Hessian-grid qparams, partial BoA for Q/K only, missing ordering heuristics, and the earlier bits-wiring bug.

V. Discussion

A. Why the gap to the BoA paper exists

We identify the following contributors:

1) **Quantization grid choice:** Hessian-weighted grid search (authors) vs MinMax (our impl) matters a lot at 2–3 bits.
2) **Partial vs full BoA:** not applying BoA to V/out and omitting exact Kronecker updates reduces the attention-specific reconstruction advantage.
3) **Act-order heuristics and best-of-two runs:** selecting the best run between different ordering variants improves low-bit results in reported experiments.
4) **Implementation bugs:** the bits-wiring issue inflated earlier experimental confusion.

B. Engineering trade-offs

The simplified choices made the method runnable on a single P100 on Kaggle, but cost final quality. We traded off fidelity to the full algorithm for practical resource constraints.

VI. Conclusions

We implemented a practical, memory-friendly variant of BoA and evaluated it on OPT-125M. Our variant recovers much of the GPTQ performance and provides some benefits at 4 bits, but fails catastrophically at 2 bits for PPL due to a combination of simplified quantization grids and partial BoA. To fully match the paper's INT3 numbers, the missing ingredients (Hessian-based grid search, BoA for V/out, ordering heuristics, correct per-quantizer bit wiring) should be implemented and tested.

VII. Future work

- Implement Hessian-weighted grid search for qparams ("Hessian" qparam_comput).
- Extend BoA to V and W_out and implement full Kronecker updates (multiply by both U_{row} and the analogous column factor).
- Add act_order row/col heuristics and run best-of-two evaluations.
- Generalize head count and remove hard-coded model-specific assumptions.
- Re-run INT2/INT3/INT4 ablations for a proper comparison with the official BoA paper.

Acknowledgements

We ran experiments in the Kaggle environment using a single NVIDIA P100 GPU. We thank the BoA authors for releasing their paper and code, and the GPTQ authors for open-source implementations that inspired the baseline.

References

[1] J. Kim, H.-y. Kim, E. Cho, C. Lee, J. Kim, and Y. Jeon, "Attention-aware post-training quantization without backpropagation," *arXiv preprint arXiv:2406.13474*, 2024.

[2] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022. (ICLR 2023)

[3] S. Zhang *et al.*, "OPT: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[4] T. Wolf, L. Debut, V. Sanh, *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020.