A Additional Experiment Results

In this section, we show additional experiment results beyond Tab. 2. Tab. 6 shows the results of RoBERTa-L finetuning on each task in GLUE datasets. Tab. 7 shows the results of GPT-2 Medium finetuning on E2E-NLG via different metrics. Tab. 8 shows the EM and F1 of RoBERTa-L finetuning on SQuAD and SQuAD 2.0 datasets.

Table 6: Performance of RoBERTa-Large finetuning on GLUE with diverse optimizers. Medians and std over 5 runs are reported on all tasks.

Optimizer	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B
32-bit AdamW	90.2 ± 0.00	94.9 ± 0.00	92.2 ± 0.00	85.2 ± 0.14	96.3 ± 0.00	93.2 ± 0.01	66.9 ± 0.01	92.3 ± 0.00
32-bit Adafactor	90.4 ± 0.00	94.7 ± 0.00	92.2 ± 0.00	85.9 ± 0.02	96.3 ± 0.00	92.8 ± 0.00	67.3 ± 0.01	92.3 ± 0.00
32-bit Adafactor‡	90.5 ± 0.00	94.8 ± 0.00	92.2 ± 0.00	87.0 ± 0.03	96.3 ± 0.00	92.9 ± 0.00	68.2 ± 0.01	92.2 ± 0.00
32-bit SM3	90.6 ± 0.00	94.2 ± 0.00	89.5 ± 0.00	85.2 ± 0.02	96.0 ± 0.00	90.5 ± 0.01	62.3 ± 0.04	91.4 ± 0.01
8-bit AdamW†	90.4 ± 0.00	94.8 ± 0.00	92.2 ± 0.00	84.8 ± 0.02	96.2 ± 0.00	93.2 ± 0.00	68.0 ± 0.00	92.2 ± 0.00
4-bit AdamW	90.2 ± 0.00	94.5 ± 0.00	92.0 ± 0.00	85.2 ± 0.12	96.3 ± 0.00	92.8 ± 0.00	67.3 ± 0.01	92.5 ± 0.00
4-bit Factor	90.1 ± 0.00	94.7 ± 0.00	92.2 ± 0.00	85.9 ± 0.00	96.4 ± 0.00	92.7 ± 0.00	68.1 ± 0.00	92.3 ± 0.00

Table 7: Performance of GPT-2 Medium finetuning on E2E-NLG Challenge with diverse optimizers. Means and std over 3 runs are reported.

Optimizer	BLEU	NIST	METEOR	ROUGE-L	CIDEr
32-bit AdamW	67.7 ± 0.67	8.60 ± 0.08	45.7 ± 0.28	68.7 ± 0.61	2.35 ± 0.04
32-bit Adafactor	67.2 ± 0.81	8.61 ± 0.60	45.3 ± 0.08	68.3 ± 0.22	2.35 ± 0.01
32-bit Adafactor‡	67.2 ± 0.63	8.54 ± 0.09	45.6 ± 0.32	68.5 ± 0.30	2.32 ± 0.02
32-bit SM3	66.9 ± 0.58	8.59 ± 0.04	45.4 ± 0.32	68.2 ± 0.49	2.33 ± 0.03
8-bit AdamW†	67.5 ± 0.87	8.59 ± 0.08	45.7 ± 0.52	68.7 ± 0.97	2.34 ± 0.06
4-bit AdamW	67.8 ± 0.51	8.61 ± 0.08	45.8 ± 0.23	68.9 ± 0.33	2.35 ± 0.07
4-bit Factor	67.6 ± 0.33	8.59 ± 0.03	45.6 ± 0.43	68.6 ± 0.60	2.34 ± 0.06

Table 8: Performance of RoBERTa-Large on SQuAD and SQuAD 2.0 with diverse optimizers. Medians and std over 5 runs are reported.

	SQuAD		SQuAD 2.0	
Optimizer	EM	F1	EM	F1
32-bit AdamW	89.0 ± 0.10	94.6 ± 0.13	85.8 ± 0.18	88.8 ± 0.15
32-bit Adafactor	88.8 ± 0.12	94.6 ± 0.14	85.8 ± 0.44	88.7 ± 0.21
32-bit Adafactor‡	89.0 ± 0.18	94.7 ± 0.10	85.9 ± 0.15	88.8 ± 0.15
32-bit SM3	84.2 ± 0.49	91.7 ± 0.29	77.2 ± 0.71	81.1 ± 0.66
8-bit AdamW†	88.8 ± 0.15	94.5 ± 0.04	86.1 ± 0.26	89.0 ± 0.26
4-bit AdamW	88.8 ± 0.08	94.5 ± 0.10	85.4 ± 0.28	88.4 ± 0.26
4-bit Factor	88.8 ± 0.38	94.6 ± 0.20	85.9 ± 0.36	88.9 ± 0.18

B Outlier Patterns of Moment

In this section, we give a comprehensive visualization about the outlier pattern of optimizer states. [2] did similar analysis for Adagrad's second-order statistics but here we give a better demonstration about various patterns in optimizer states. The same technique has been applied to parameters and activations in [50].

The outlier pattern of first-order momentum depends on many factors such as data and training hyperparameters. Here we mainly focus on different transformer models and different layers inside. In one transformer block, there is one Attention module and one MLP module, including 6 main parameter matrices. We do not focus on additional parameters including bias and parameters in layer normalization (LN) since they only account a small portion. We denote the 6 matrices by $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V, \mathbf{W}^O, \mathbf{W}^1, \mathbf{W}^2$, respectively. When the matrices across different layers are involved at the same time, we add a subscript indicating layer index. Note that $\mathbf{W}$ has shape $\mathbb{R}^{C_\text{o} \times C_\text{i}}$ in a linear layer, we call the output and input dimension by dim0/row and dim1/column, respectively. The per-channel quantizaition used in other works actually correspond to per-row(dim0) normalization here.

Swin Transformer ImageNet pretraining In Fig. 5,6,7, the magnitude of first-order momentum in transformer blocks at different depths are shown. It can be seen that the 1-dimensional structure in all parameter matrices are vague at the initial layer. At layer 2, the pattern in $\mathbf{W}^O$ and $\mathbf{W}^1$, that outliers occur at fixed columns, becomes obvious while other parameter matrices remain noisy. At layer 3, the patterns in $\mathbf{W}^O, \mathbf{W}^1, \mathbf{W}^2$ are quite obvious. In $\mathbf{W}^O$ and $\mathbf{W}^2$, the outliers occur at fixed rows while the pattern in $\mathbf{W}^1$ remain unchanged. The 1-dimensional structure in $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$ also seem to appear even though not remarkable. It is notable that the pattern of same parameter matrix at different depths are not necessarily same. See $\mathbf{W}^O$ in Fig. 6,7.

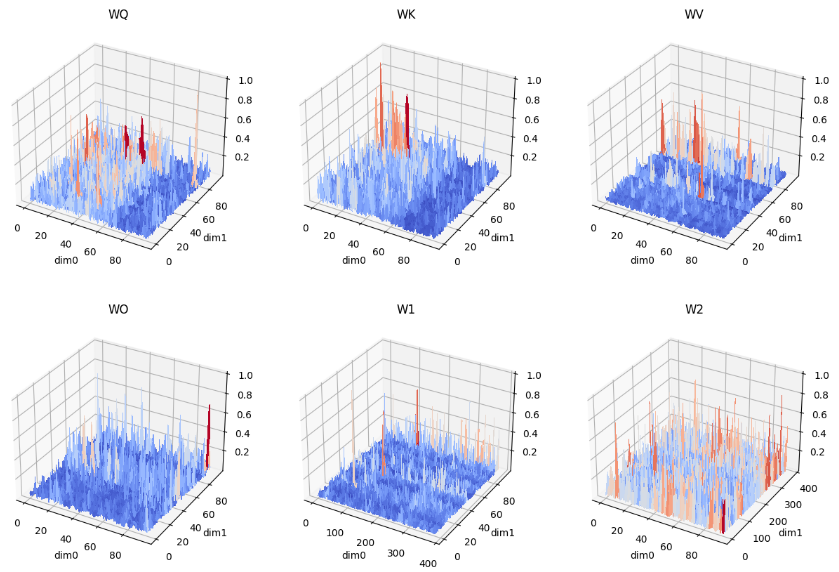

Figure 5: Outlier patterns of first moment in transformer block `layers.0.blocks.0` of Swin-T at epoch 210.

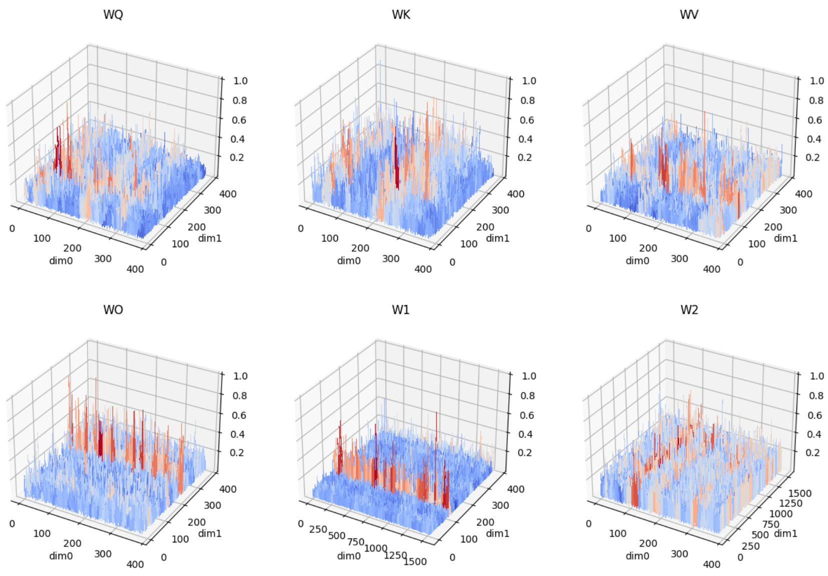

Figure 6: Outlier patterns of first moment in transformer block `layers.2.blocks.0` of Swin-T at epoch 210.

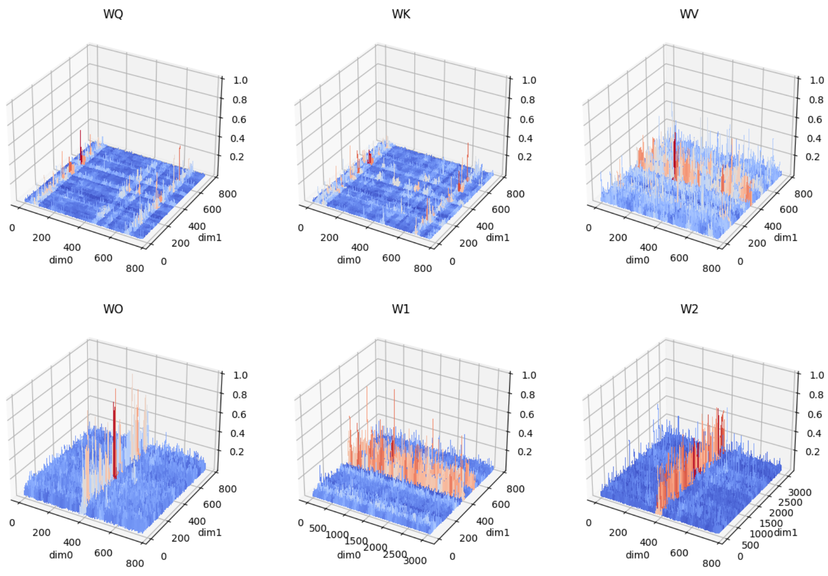

Figure 7: Outlier patterns of first moment in transformer block `layers.3.blocks.0` of Swin-T at epoch 210.

RoBERTa-Large GLUE finetuning In Fig. 8,9,10,11,12,13, the magnitude of first-order momentum in transformer blocks of RoBERTa-Large at different depths are shown. At layer 0 and layer 1 (initial layers), patterns in $\mathbf{W}^O, \mathbf{W}^2$ are obvious. At layer 11 and layer 12 (intermediate layers), patterns are all noisy. At layer 22 and layer 23 (last layers), patterns in $\mathbf{W}^Q, \mathbf{W}^K$ are obvious. Patterns in other matrices are weak.

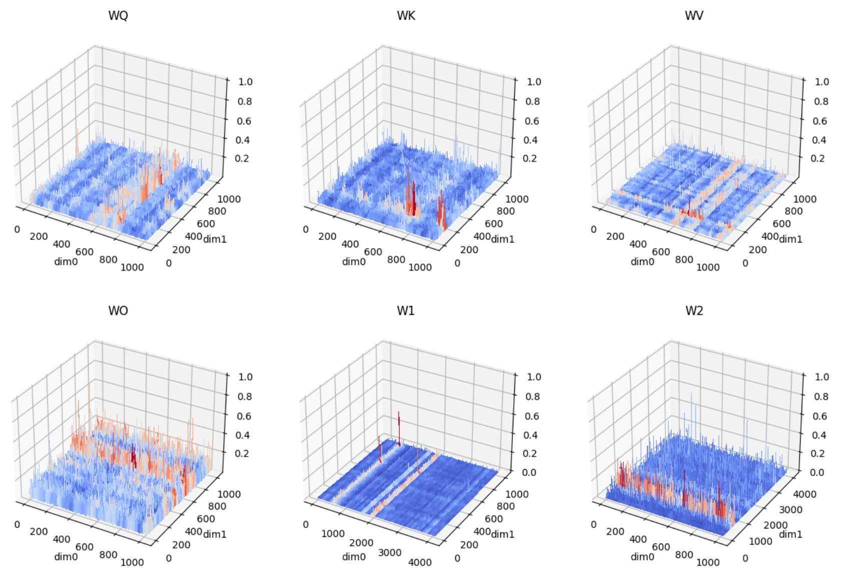

Figure 8: Outlier patterns of first moment in transformer block layer-0 of RoBERTa-Large at epoch 8.

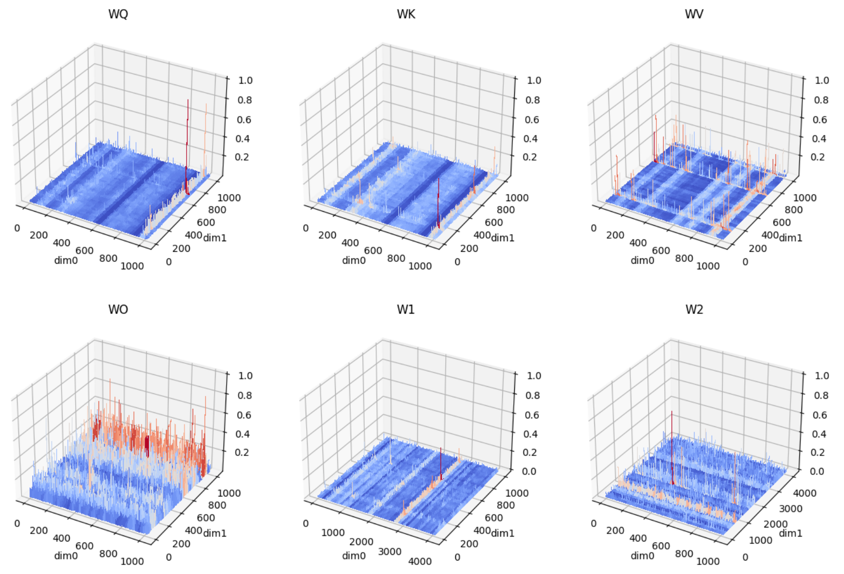

Figure 9: Outlier patterns of first moment in transformer block layer-1 of RoBERTa-Large at epoch 8.

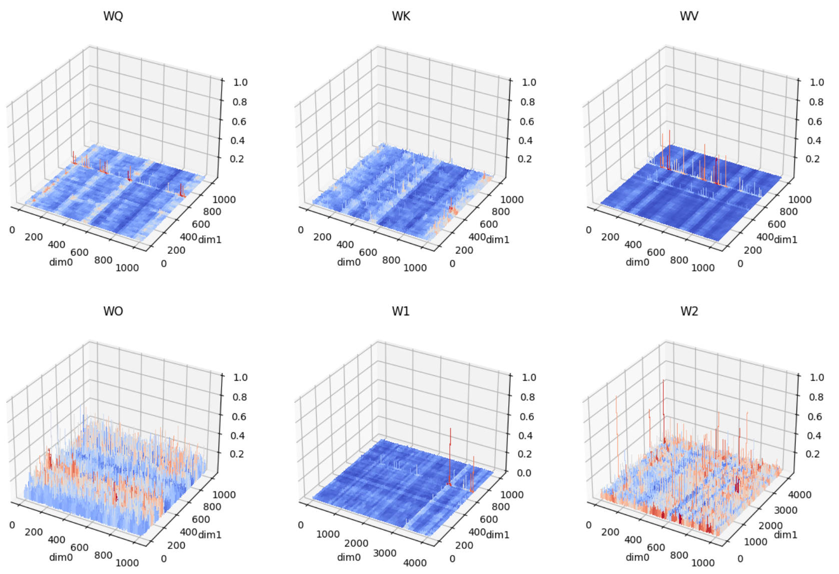

Figure 10: Outlier patterns of first moment in transformer block layer-11 of RoBERTa-Large at epoch 8.

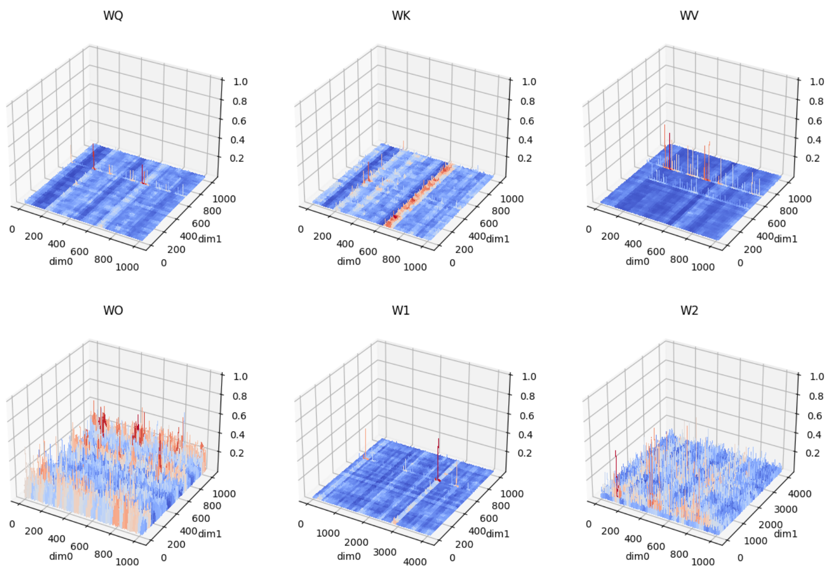

Figure 11: Outlier patterns of first moment in transformer block layer-12 of RoBERTa-Large at epoch 8.

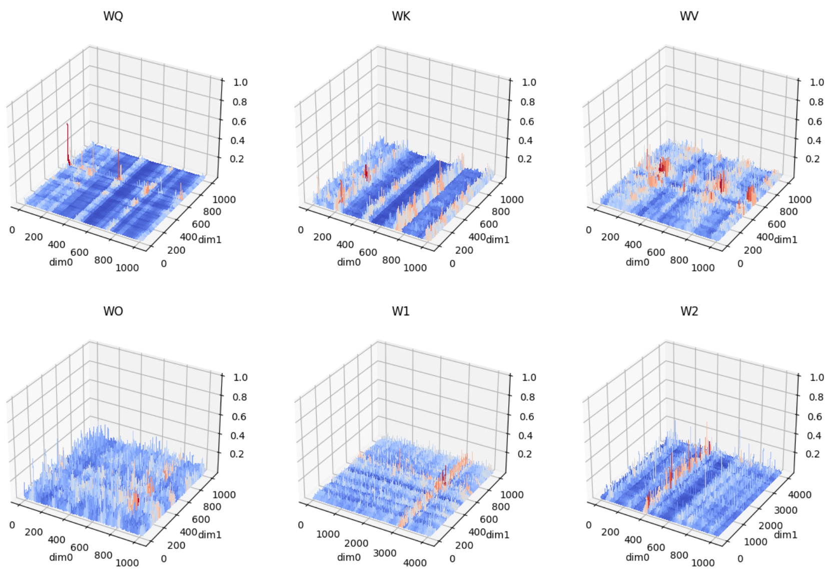

Figure 12: Outlier patterns of first moment in transformer block layer-22 of RoBERTa-Large at epoch 8.

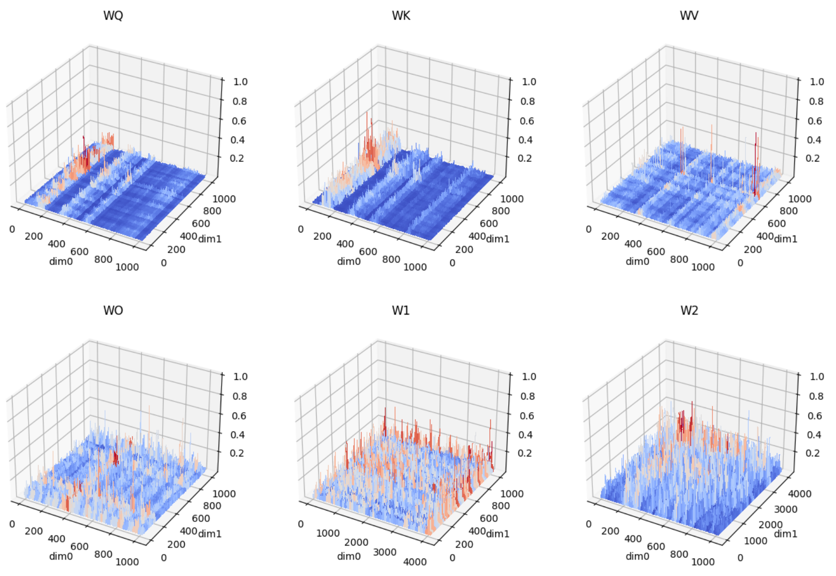

Figure 13: Outlier patterns of first moment in transformer block layer-23 of RoBERTa-Large at epoch 8.

GPT-2 Medium E2E-NLG finetuning In Fig. 14,15,16,17,18,19, the magnitude of first-order momentum in transformer blocks of GPT-2 Medium at different depths are shown. At layer 1 and layer 2 (initial layers), patterns in $\mathbf{W}^O$ are obvious. At layer 13 and layer 14 (intermediate layers), patterns in $\mathbf{W}^K$, $\mathbf{W}^O$ are obvious. At layer 21 and layer 22 (last layers), patterns in $\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V$, $\mathbf{W}^O$ are obvious. First-order momentum of $\mathbf{W}^1$, $\mathbf{W}^2$ are consistently noisy throughout layers. It is notable that the rows(or columns) that gather outliers are different across different layers.

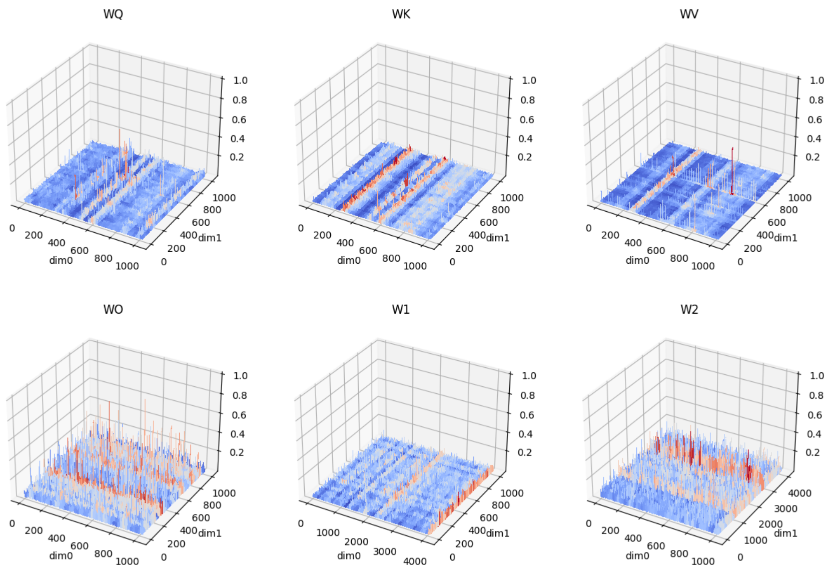

Figure 14: Outlier patterns of first moment in transformer block layer-1 of GPT-2 Medium at epoch 2.

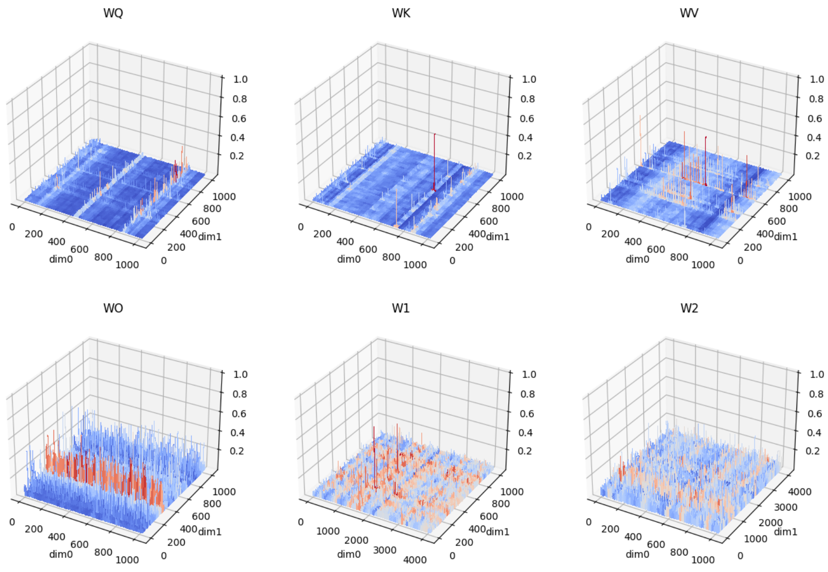

Figure 15: Outlier patterns of first moment in transformer block layer-2 of GPT-2 Medium at epoch 2.

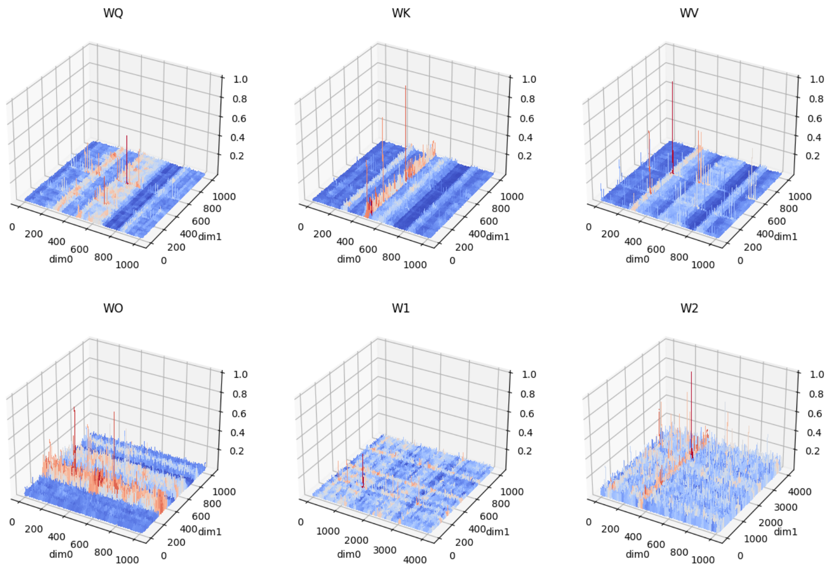

Figure 16: Outlier patterns of first moment in transformer block layer-13 of GPT-2 Medium at epoch 2.

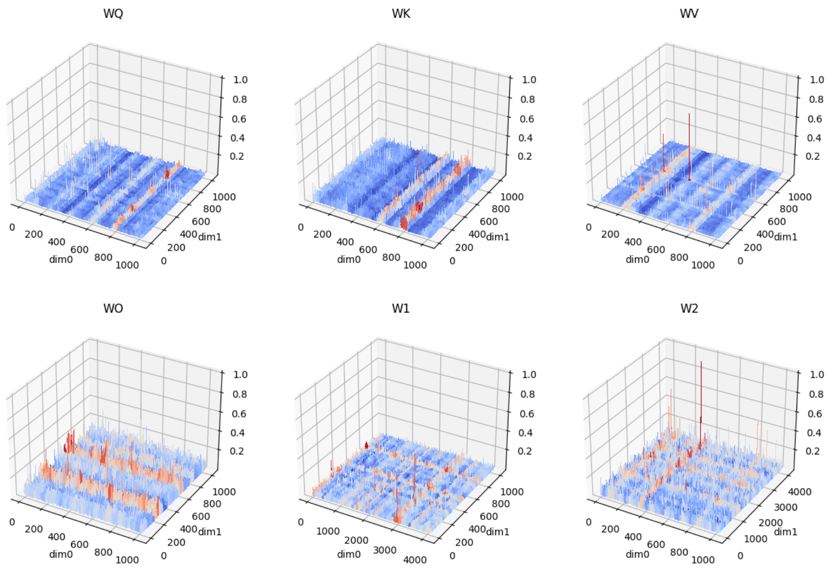

Figure 17: Outlier patterns of first moment in transformer block layer-14 of GPT-2 Medium at epoch 2.

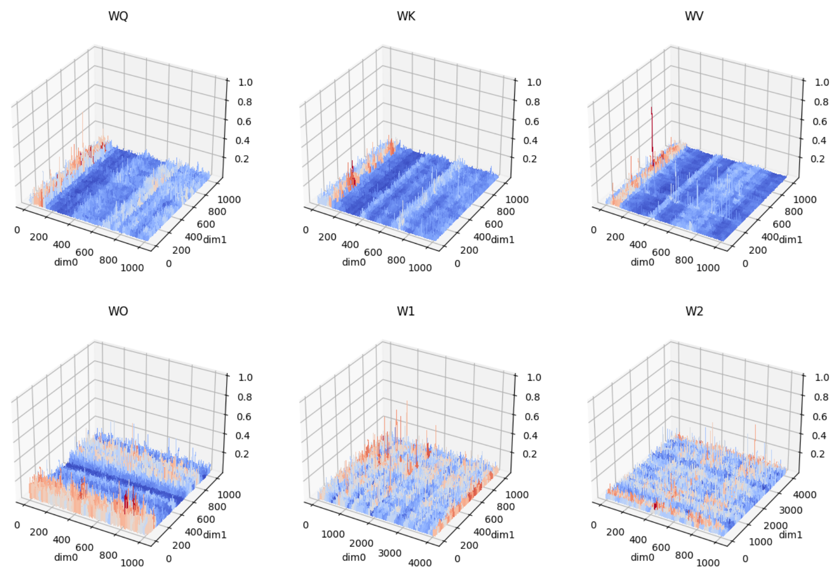

Figure 18: Outlier patterns of first moment in transformer block layer-21 of GPT-2 Medium at epoch 2.

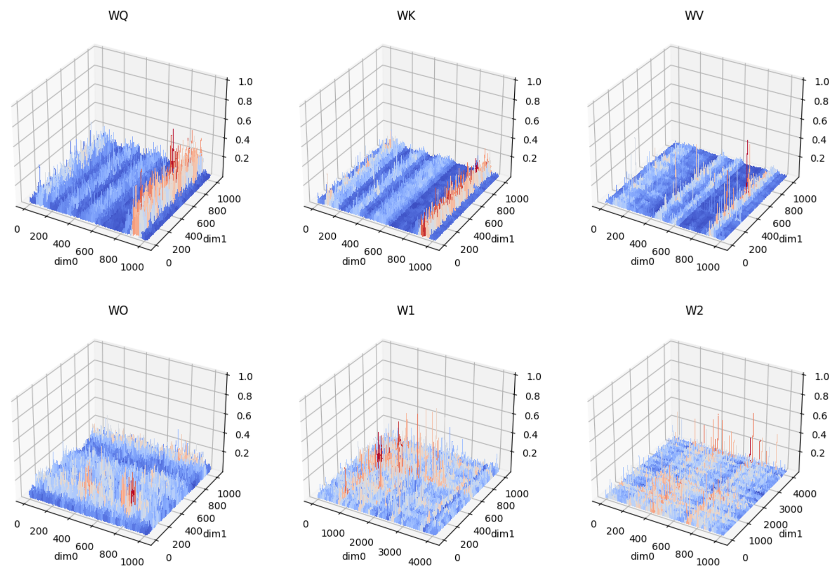

Figure 19: Outlier patterns of first moment in transformer block layer-22 of GPT-2 Medium at epoch 2.

C Quantization Quality via Histogram

C.1 Zero-point Problem

In Fig. 20,21,22, we show the effect of zero-point on quantization error for second-order momentum via histogram. All those figures show the negative impact of zero-point on quantizing second-order momentum. After removing zero-point, the quantization quality improves at a great scale.

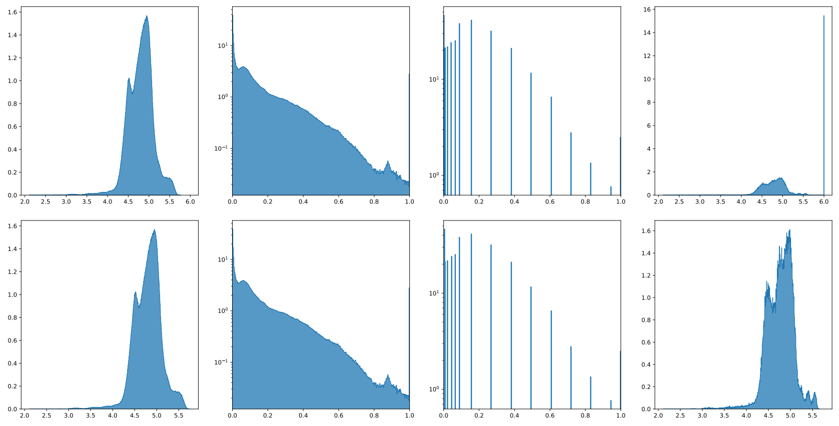

Figure 20: Histogram of second-order momentum of the attention layer ($\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V$) in transformer block-wise layer-2 of GPT-2 Medium at epoch 2. In one horizontal line, the first figure is the original second-order momentum. The second figure is the tensor after normalization. The third figure is the quantized tensor. The last figure is the dequantized object. Both the first and last figure is at log10 scale. Both the second and third take values in [0, 1]. All y-axis represents density. Good quantization methods try to make the third figure identical to the second figure and make the last figure identical to the first figure. Top: B128/DE quantization. Bottom: B128/DE-0 quantization.

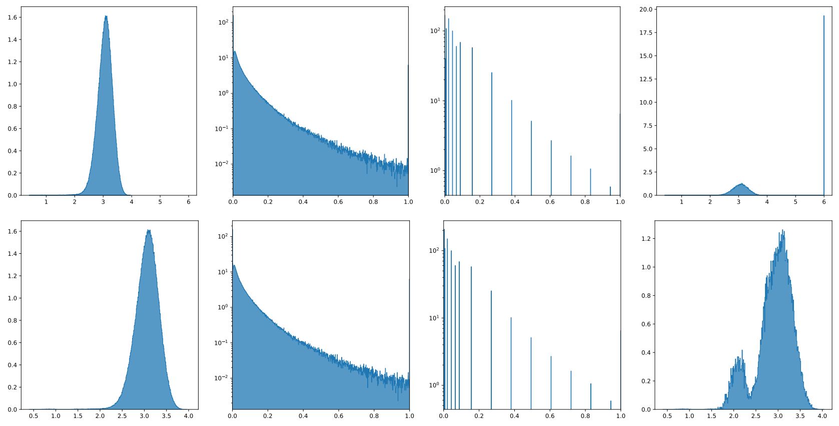

Figure 21: Histogram of second-order momentum of the $\mathbf{W}^V$ in transformer block layer-10 of RoBERTa-Large at epoch 8. Top: B128/DE quantization. Bottom: B128/DE-0 quantization.

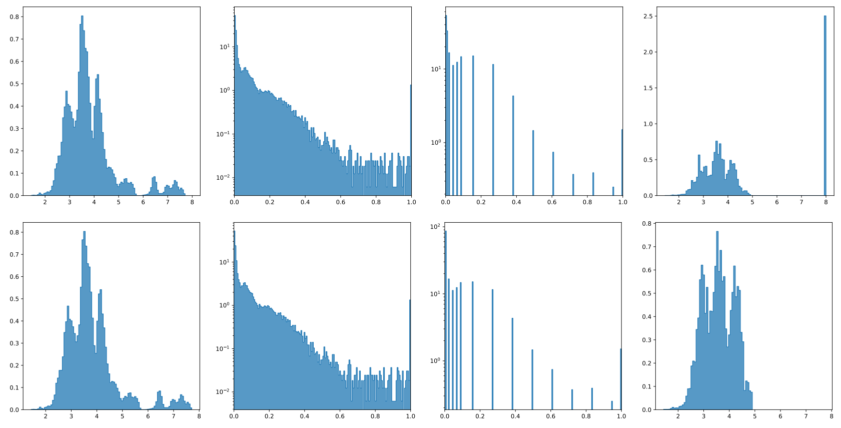
Figure 22: Histogram of second-order momentum of the attention layer ($\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V$) in transformer block `layers.0.blocks.0` of Swin-T at epoch 210. Top: B128/DE quantization. Bottom: B128/DE-0 quantization.

C.2 Comparison between Block-wise and Rank-1 Normalization

To show the differences in quantization error for second-order momentum between block-wise normalization and rank-1 normalization, some cases where rank-1 normalization approximates better than block-wise normalization are shown in Fig. 23, 25, 27. Also, some cases where rank-1 normalization approximates worse than block-wise normalization is shown in Fig. 24, 26, 28. Empirically, it has been observed that rank-1 normalization yields superior results when the distribution exhibits long-distance multimodal characteristics. On the other hand, block-wise normalization tends to outperform when the distribution displays short-distance multimodal patterns and/or intricate local structures.

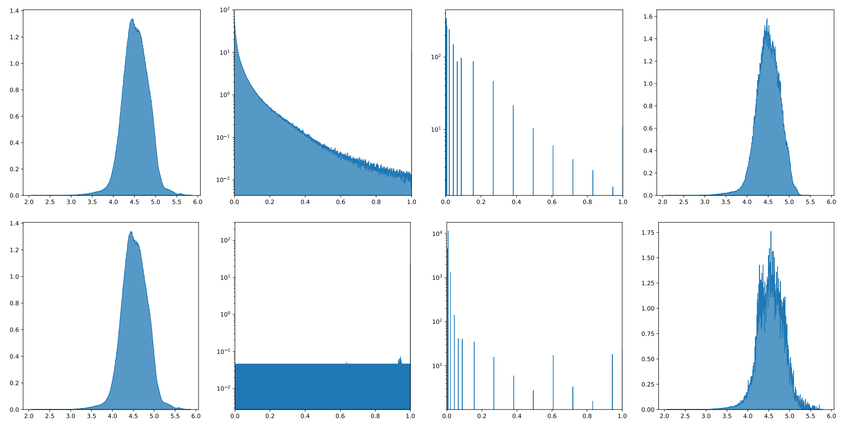
Figure 23: Histogram of second-order momentum of $\mathbf{W}^1$ in transformer block layer-23 of GPT-2 Medium at epoch 2. A case where rank-1 normalization is better than block-wise normalization with block size 128. In this case, the tail in the right side of distribution is captured by rank-1 normalization but lost in block-wise normalization. Top: B128/DE-0 quantization. Bottom: Rank-1/DE-0 quantization.

C.3 Effectiveness of Block Size in Block-wise Normalization

In Fig. 29,30,31, we show the effect of block size on quantization error for both first-order momentum and second-order momentum. Fig. 29,30 shows that B2048 normalization quantizes a significant portion of the points to zero, resulting in poor approximation based on the histogram. However, when

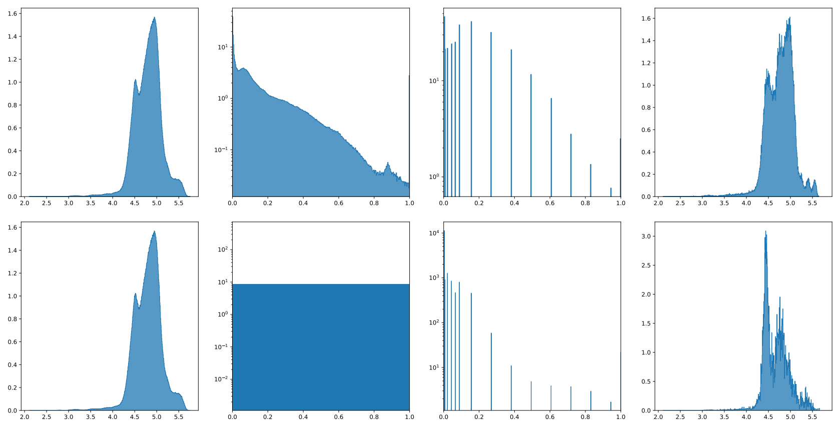

Figure 24: Histogram of second-order momentum of the attention layer ($\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$) in transformer block layer-2 of GPT-2 Medium at epoch 2. A case where rank-1 normalization is worse than block-wise normalization with block size 128. Top: B128/DE-0 quantization. Bottom: Rank-1/DE-0 quantization.

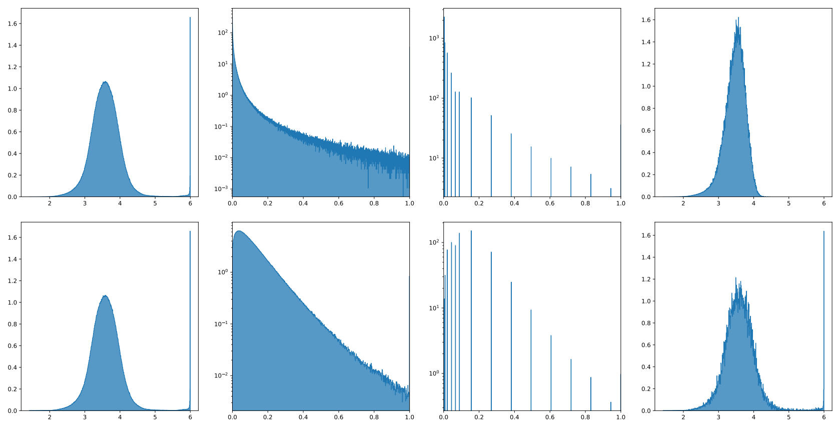

Figure 25: Histogram of second-order momentum of $\mathbf{W}^2$ in transformer block layer-4 of RoBERTa-Large at epoch 8. A case where rank-1 normalization is better than block-wise normalization with block size 128. Top: B128/DE-0 quantization. Bottom: Rank-1/DE-0 quantization.

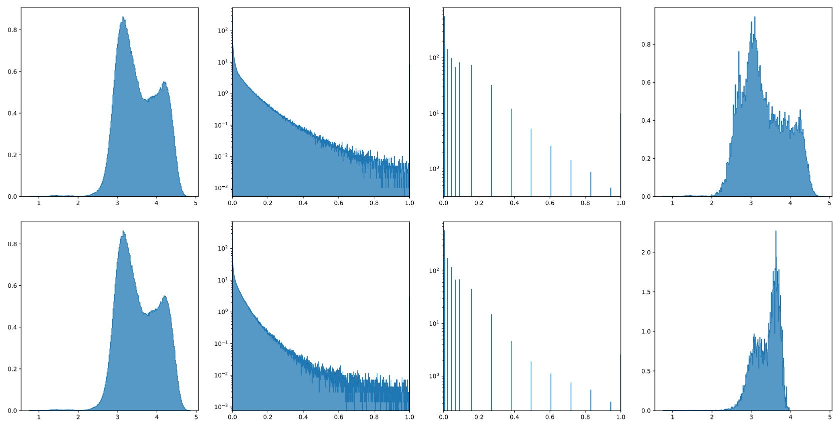

Figure 26: Histogram of second-order momentum of $\mathbf{W}^V$ in transformer block layer-2 of RoBERTa-Large at epoch 8. A case where rank-1 normalization is worse than block-wise normalization with block size 128. Top: B128/DE-0 quantization. Bottom: Rank-1/DE-0 quantization.

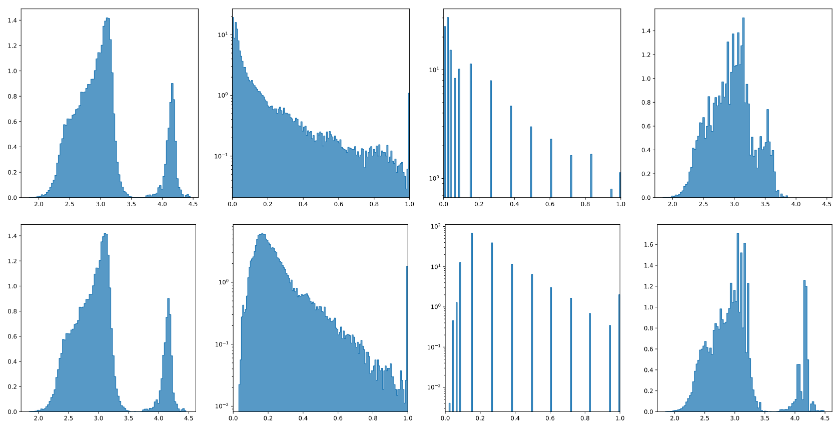
Figure 27: Histogram of second-order momentum of $\mathbf{W}^2$ in transformer block `layers.0.blocks.0` of Swin-T at epoch 210. A case where rank-1 normalization is better than block-wise normalization with block size 128. Top: B128/DE-0 quantization. Bottom: Rank-1/DE-0 quantization.

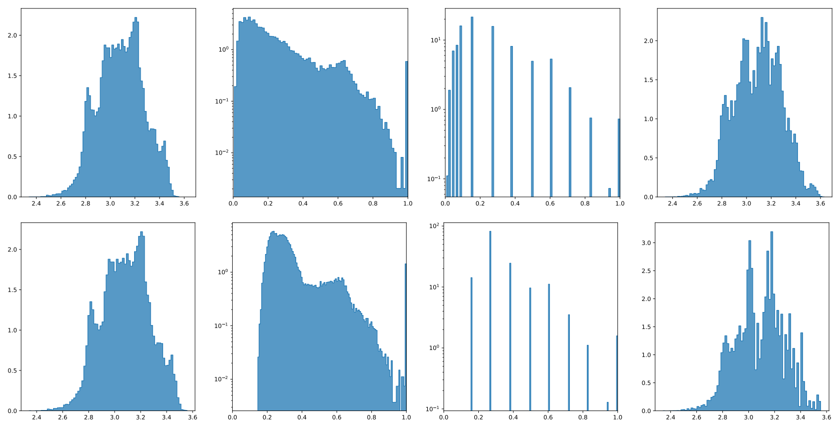
Figure 28: Histogram of second-order momentum of $\mathbf{W}^O$ in transformer block `layers.1.blocks.0` of Swin-T at epoch 210. A case where rank-1 normalization is worse than block-wise normalization with block size 128. Top: B128/DE-0 quantization. Bottom: Rank-1/DE-0 quantization.

we utilize a smaller block size of 128, the quantization performance improves. Fig. 31 shows smaller block size improves quantization quality on second-order momentum.

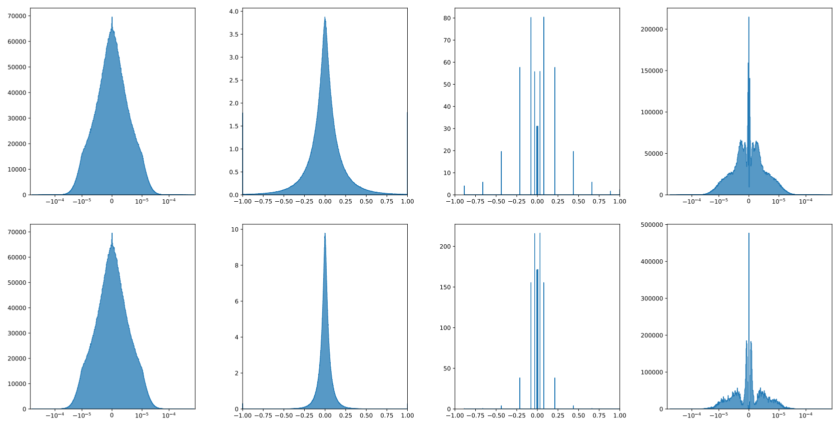

Figure 29: Histogram of first-order momentum of $\mathbf{W}^1$ in transformer block layer-20 of GPT-2 Medium at epoch 2. Top: B128/DE quantization. Bottom: B2048/DE quantization.

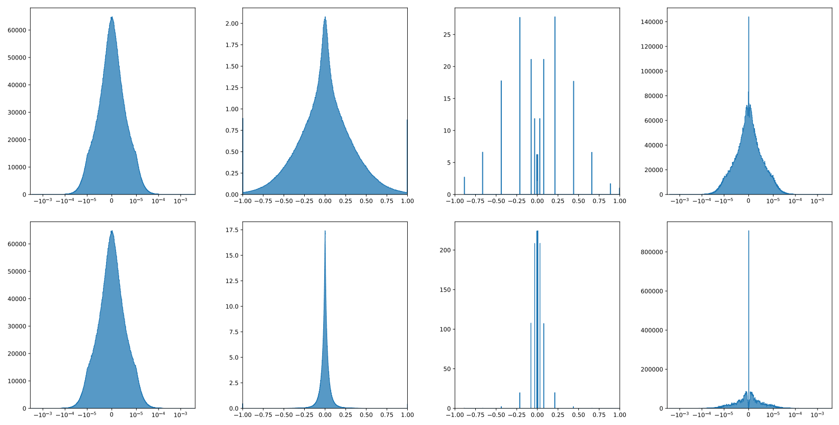

Figure 30: Histogram of first-order momentum of $\mathbf{W}^O$ in transformer block layer-22 of RoBERTa-L at epoch 8. Top: B128/DE quantization. Bottom: B2048/DE quantization.

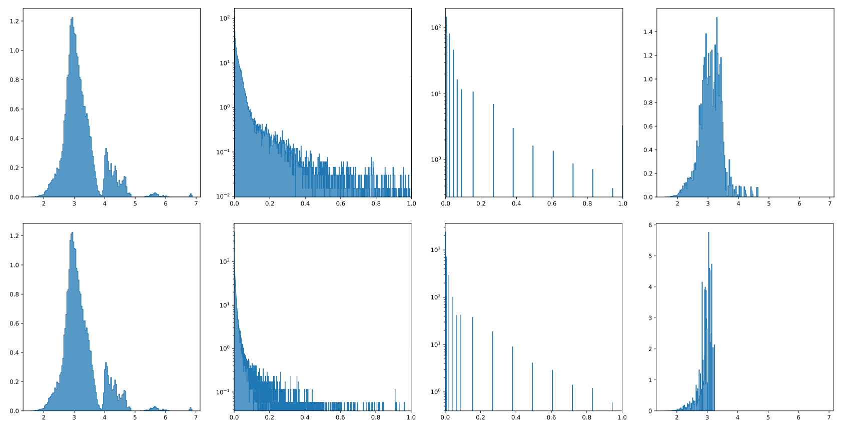

Figure 31: Histogram of second-order momentum of $\mathbf{W}^1$ in transformer block `layers.0.blocks.0` of Swin-T at epoch 210. Top: B128/DE-0 quantization. Bottom: B2048/DE-0 quantization.

D Experimental Details

D.1 Quantization

There are several parameters in deep neural networks that play a delicate role without occupying too much memory, such as normalization layers and bias. In this context, we establish a rule to determine which parameters should not be quantized. For all the experiments we conducted, the rule is straightforward: tensors with a size smaller than or equal to 4096 will not be quantized. However, for larger models with a hidden size exceeding 4096, it is advisable to exclude the bias and normalization layers from quantization. Regarding the quantization settings, as stated in Sec. 5, we employ block-wise normalization with a block size of 128, dynamic exponent mapping for first-order momentum and rank-1 normalization, linear mapping for second-order momentum. When we apply factorization on second-order momentum, only tensors with a dimension greater than or equal to 2 will be factorized while 1-dimensional tensors that meet the specified rule will still be quantized.

8-bit Adam [14] also uses the threshold of 4096 about size to determine whether or not to quantize parameters. Additionally, the implementation in huggingface does not quantize the parameters in `Embedding` layers regardless of the model used. Consequently, we compare our method with the 8-bit Adam that does not quantize `Embedding`.

D.2 Hyperparameters and Training Details

In each benchmark, unless otherwise specified, we maintain the same hyperparameters for a given optimize across different quantization schemes. Additionally, we use same optimizer hyperparameters across various optimizers, including our 4-bit optimizers, 8-bit Adam [14], SM3 [2], Adafactor [42] and the full precision counterpart AdamW [30]. For Adafactor, we use $\beta_1 > 0$ as default setting which is same as the β_1 value used in AdamW. Also, the case where $\beta_1 = 0$ is compared. The other newly introduced hyperparameters in Adafactor are set to their default values and remain fixed throughout the experiments. For SM3, we compare with the $\beta_1 > 0$ configuration, same as the β_1 value used in AdamW.

Table 9: The hyperparameters for RoBERTa-L finetuning on GLUE.

Dataset	MNLI	QNLI	QQP	RTE	MRPC	SST-2	CoLA	STS-B
Batch Size	32	32	32	16	16	32	16	16
LR	1e-5	1e-5	1e-5	2e-5	1e-5	1e-5	1e-5	2e-5
Warmup	7432	1986	28318	122	137	1256	320	214
Max Train Steps	123873	33112	113272	2036	2296	20935	5336	3598
Max Seq. Len.	128	128	128	512	512	512	512	512

Table 10: The hyperparameters for RoBERTa-L finetuning on SQuAD and SQuAD 2.0.

Dataset	SQuAD & SQuAD 2.0
Batch Size	48
LR	1.5e-5
# Epochs	2
Warmup Ratio	0.06
Max Seq. Len.	384

RoBERTa We train all of our RoBERTa-L models with PyTorch Huggingface[7]. On GLUE benchmark, we mainly follow the hyperparameters in fairseq [32]. We use $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon =$ 1e-6, a weight decay factor of 0.1 and linear learning rate schedule. Other hyperparameters are listed in Tab. 9. On SQuAD benchmark, we mainly follow the reported hyperparameters in RoBERTa paper [28]. We use $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon =$ 1e-6, a weight decay factor of 0.01 and linear learning rate schedule. The other hyperparameters are listed in Tab. 10. On both datasets, we report the median

[7] `https://github.com/huggingface/transformers`

Table 11: The hyperparameters for GPT-2 on E2E.

Dataset	E2E
	Training
Batch Size	8
LR	4e-5
# Epochs	5
Warmup	500
Max Seq. Len.	512
Label Smooth	0.1
	Inference
Beam Size	10
Length Penalty	0.8
no repeat ngram size	4

and standard deviation results over 5 runs, the result in each run is taken from the best epoch. We utilize single RTX 3090 or 4090 GPU for runs of each task in GLUE datasets and four RTX 3090 or 4090 GPUs for SQuAD and SQuAD 2.0.

On SQuAD 2.0, there may be a performance gap observed between the reproduced results using 32-bit AdamW and the original results reported in the original paper. This is because there are some questions without answers in SQuAD 2.0. It is worth noting that the approach employed by Liu et al. [28] to handle unanswered questions may differ from the solutions utilized in the BERT paper [16], which is the reference implementation we are using

GPT-2 We train all of our GPT-2 Medium models with the LoRA codebase[8]. We mainly follow the hyperparameters in [26] and [22]. We use $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon =$ 1e-6, a weight decay factor of 0.01 and linear learning rate schedule. The other hyperparameters used in GPT-2 are listed in Tab. 11. We report the mean and standard deviation results over 3 runs, the result in each run is taken from the best epoch. We utilize fours RTX 3090 or 4090 GPUs for runs of this task.

Transformer We train all of our Transformer-Base models for machine translation with codebase[9]. We completely follow the hyperparameters in the codebase. We report the mean and standard deviation results over 3 runs, the result in each run is taken from the best epoch. We utilize eight RTX 3090 or 4090 GPUs for runs of this task.

Swin We train all of our Swin-T models with its official codebase[10]. We completely follow the hyperparameters in the codebase. We report the mean and standard deviation results over 3 runs, the result in each run is taken from the best epoch. We utilize eight RTX 3090 or 4090 GPUs for runs of this task.

LLaMA We finetune LLaMA-7B with Alpaca codebase[11]. We follow the hyperparameters in the codebase except the we finetune the model using two A100 80GB GPUs. The training loss curve is the mean results over 3 runs. For the LLaMA-7B model, we enable Fully Sharded Data Parallelism (FSDP), which packs parameters into 1-dimensional array. This packing process makes it difficult to apply factorization directly without additional engineering efforts. Consequently, we only compare the performance of 4-bit AdamW with its full precision counterpart.

D.3 Memory and Computing Efficiency

In Tab. 3, we present measurements of memory usage in practical settings, i.e. training configuration described in Sec. D.2. Specifically, we measure the memory usage for LLaMA-7B using 2 A100

[8] https://github.com/microsoft/LoRA

[9] https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Translation/Transformer

[10] https://github.com/microsoft/Swin-Transformer

[11] https://github.com/tatsu-lab/stanford_alpaca

80G GPUs, RoBERTa-L using 1 RTX 4090 GPU, and GPT-2 Medium using 4 RTX 4090 GPUs. Additionally, the time measurement for RoBERTa-L is conducted on the RTE task.

E Quantization Formulation Details

E.1 Signed Case

In this section, we discuss quantization function for signed tensors and the differences compared to unsigned case. Regarding the normalization operator, the only difference lies in the fact that the sign of the tensor remains unchanged before and after normalization. Formally, let $\mathbf{N}$ be the normalization operator for the unsigned cases. For the signed case, the normalization can be defined as

$$n_j := \text{sign}(x_j)\mathbf{N}(|x_j|).$$

Therefore, the unit interval for signed case is [-1, 1]. Regarding the mapping operator, the difference lies in the values of quantization mappings. See App. E.2 for more details.

E.2 Quantization Mappings

In this work, we mainly consider linear mapping and dynamic exponent mapping [12]. See Fig. 32 for illustration of quantization mappings.

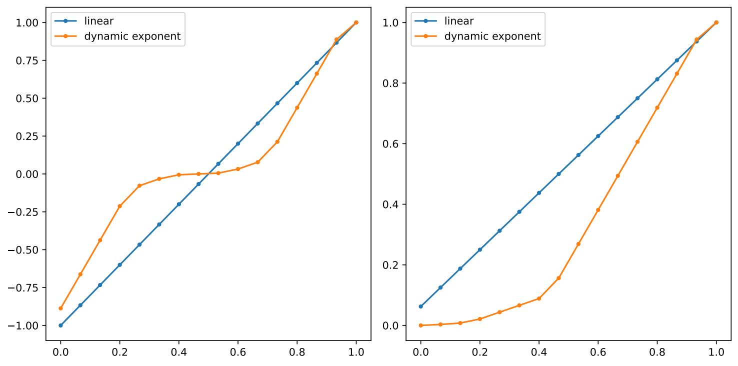

Figure 32: Visualization of the quantization mappings for the linear and dynamic exponent at 4-bit precision. Left: Signed case. Right: Unsigned case.

Linear mapping It is notable that the linear mapping considered in our work does not include zero in both signed case and unsigned case. Actually, we only use linear mapping in unsigned case, which is defined as `torch.linspace(0, 1, (2 ** b) + 1)[1:]`.

Dynamic exponent mapping Let b be the total bits. In the main text, we mentioned that dynamic exponent takes the form $\mathbf{T}(i) = 10^{-E(i)}\text{fraction}(i)$. In following paragraphs, we will define the dynamic exponent mapping formally based on the binary representation.

In unsigned case, dynamic exponent mapping [12] is composed of exponent bits E, one indicator bit and fraction bits F, where $b = 1 + E + F$. It uses the number of leading zero bits E represents the exponent with base 10. The first bit, which is one, serves as an indicator bit that separates the exponent and the unsigned linear fraction. The remaining bits F represent an unsigned linear fraction distributed evenly in (0.1, 1), which is formally defined as

$$p_j = \frac{1 - 0.1}{2^F} j + 0.1, \quad 0 \le j \le 2^F,$$

$$\text{fraction}[k] = \frac{p_k + p_{k+1}}{2}, \quad 0 \le k \le 2^F - 1.$$

Therefore, a number with E exponent bits and F fraction bits valued k has a value of

$$10^{-E} \times \text{fraction}[k].$$

For signed case, the only difference is that dynamic exponent mapping additionally uses the first bit as the sign thus we have $b = 1 + E + 1 + F$. Specially, at 8-bit case, we learn from the codebase[12] that dynamic exponent mapping assign $00000000_2 = 0_{10}$, $00000001_2 = 1_{10}$ in unsigned case and assign 10000000_2 and 00000000_2 with 1_{10} and 0_{10}, respectively. This means -1_{10} is not defined and the mapping is not symmetric in signed case. Finally, after collecting all the represented numbers and arranging them in a sorted, increasing list, which has a length of 2^b, the quantization mapping $\mathbf{T}(i)$ returns the i-th element of this list.

The construction of dynamic exponent mapping is unrelated to the number of bits. Therefore, when we say we barely turn the 8-bit optimizer into 4-bit optimizer, it just use 4 total bits. The corner cases mentioned in last paragraph remain unchanged.

E.3 Stochastic Rounding

Stochastic rounding is only used in Tab. 1. In this section, we talk about how to integrate stochastic rounding into our formulation of quantization. When stochastic rounding is used, the definition of mapping $\mathbf{M}$ has some minor changes. Specifically, $\mathbf{M}$ is still an element-wise function and defined as

$$\mathbf{M}(n_j) = \arg\min_{0 \le i < 2^b} \{n_j - \mathbf{T}(i) : n_j - \mathbf{T}(i) \ge 0\} \cup \arg\max_{0 \le i < 2^b} \{n_j - \mathbf{T}(i) : n_j - \mathbf{T}(i) \le 0\}.$$

In other words, $\mathbf{M}$ maps each entry n_j to the maximal index set $\mathbf{M}(n_j)$ such that for any $i \in \mathbf{M}(n_j)$ there is no other $0 \le k \le 2^b - 1$ with $\mathbf{T}(k)$ lying between $\mathbf{T}(i)$ and n_j. Actually, $\mathbf{M}$ acts as a filter of $\mathbf{T}$ and give a more fine-grained range of quantized output candidates. In this definition, $\mathbf{M}(n_j)$ has only one or two points since only stochastic rounding is considered in the final step.

Finally, we define stochastic rounding $\mathbf{R}_s$. When $\mathbf{M}(n_j)$ only has one point, $\mathbf{R}_s$ just output this point. When $\mathbf{M}(n_j)$ has two points q_1 and q_2 with $\mathbf{T}(q_1) < n_j < \mathbf{T}(q_2)$, stochastic rounding ($\mathbf{R}_s$) is defined as

$$\mathbf{R}_s(n_j, q_1, q_2) = \begin{cases} q_2, \text{with proba. } \frac{n_j - \mathbf{T}(q_1)}{\mathbf{T}(q_2) - \mathbf{T}(q_1)} \\ \\ q_1, \text{with proba. } \frac{\mathbf{T}(q_2) - n_j}{\mathbf{T}(q_2) - \mathbf{T}(q_1)} \end{cases}$$

F Compression-based Memory Efficient Optimizer Instances

In this section, we present some examples about compression-based memory efficient optimizers. See Compression-based Memory Efficient SGDM in Alg. 2 and Adam in Alg. 3.

Algorithm 2 Compression-based Memory Efficient SGDM

```
Require: initial parameter $\theta_0 \in \mathbb{R}^p$, learning rate $\alpha$, initial momentum $\bar{m}_0 = 0$, total number of
    iterations $T$ and momentum parameter $\beta$.
1: for $t = 1, 2, \ldots, T$ do
2:     Sample a minibatch $\zeta_t$ and get stochastic gradient $g_t = \nabla_\theta f(\theta_{t-1}, \zeta_t)$
3:     $m_{t-1} \leftarrow$ decompress($\bar{m}_{t-1}$)
4:     $m_t \leftarrow \beta \cdot m_{t-1} + g_t$
5:     $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot m_t$
6:     $\bar{m}_t \leftarrow$ compress($m_t$)
7: end for
8: return $\theta_T$
```

G Rank-1 Normalization

In this section, we present the detailed formulation of rank-1 normalization in Alg. 4.

[12] https://github.com/TimDettmers/bitsandbytes

Algorithm 3 Compression-based Memory Efficient Adam

```
Require: initial parameter $\theta_0 \in \mathbb{R}^p$, learning rate $\alpha$, initial momentum $\bar{m}_0 = 0, \bar{v}_0 = 0$, total
    number of iterations $T$ and hyperparameters $\beta_1, \beta_2, \epsilon$.
 1: for $t = 1, 2, \ldots, T$ do
 2:     Sample a minibatch $\zeta_t$ and get stochastic gradient $g_t = \nabla_\theta f(\theta_{t-1}, \zeta_t)$
 3:     $m_{t-1}, v_{t-1} \leftarrow \text{decompress}(\bar{m}_{t-1}), \text{decompress}(\bar{v}_{t-1})$
 4:     $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
 5:     $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_1) \cdot g_t^2$
 6:     $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
 7:     $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
 8:     $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$
 9:     $\bar{m}_t, \bar{v}_t \leftarrow \text{compress}(m_t), \text{compress}(v_t)$
10: end for
11: return $\theta_T$
```

Algorithm 4 Rank-1 Normalization

```
Require: tensor $x \in \mathbb{R}^{d_1 \times \cdots \times d_p}$; statistics $\mu_r \in \mathbb{R}^{d_r}$ for $1 \leq r \leq p$; permutation function $\Phi$
    mapping $\{1, \ldots, d\}$ to indices of tensor $x$, where $d = d_1 \times \cdots \times d_p$.
 1: for $r = 1, 2, \ldots, p$ do
 2:     for $j = 1, 2, \ldots, d_r$ do
 3:         $\mu_{r,j} = \max_{i_1, \ldots, i_{r-1}, i_{r+1}, \ldots, i_p} \left| x_{[i_1, \ldots, i_{r-1}, j, i_{r+1}, \ldots, i_p]} \right|$
 4:     end for
 5: end for
 6: for $i = 1, 2, \ldots, d$ do
 7:     $M_i = \min_{1 \leq r \leq p} \mu_{r, \Phi(i)_r}$
 8: end for
 9: reshape 1-dimensional array $M$ to the same shape as $x$
10: return $x/M$
```