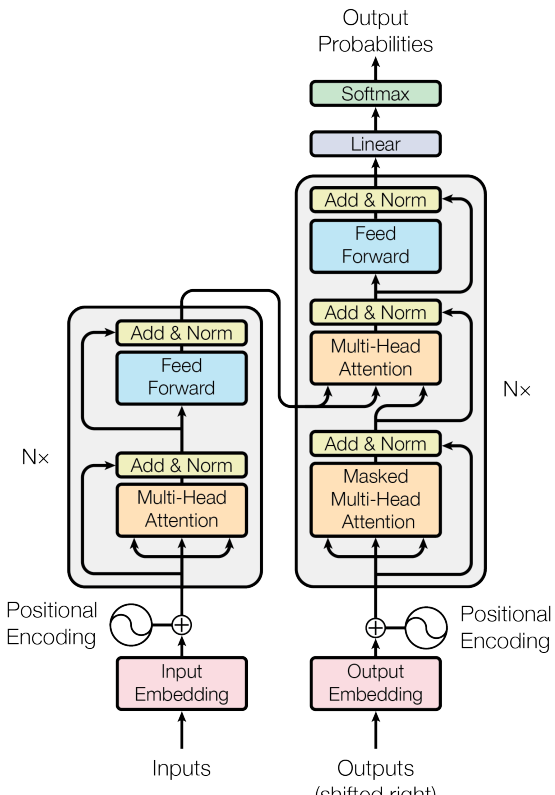
LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale

Yuxiang Qiu
2025.03.28

Outline

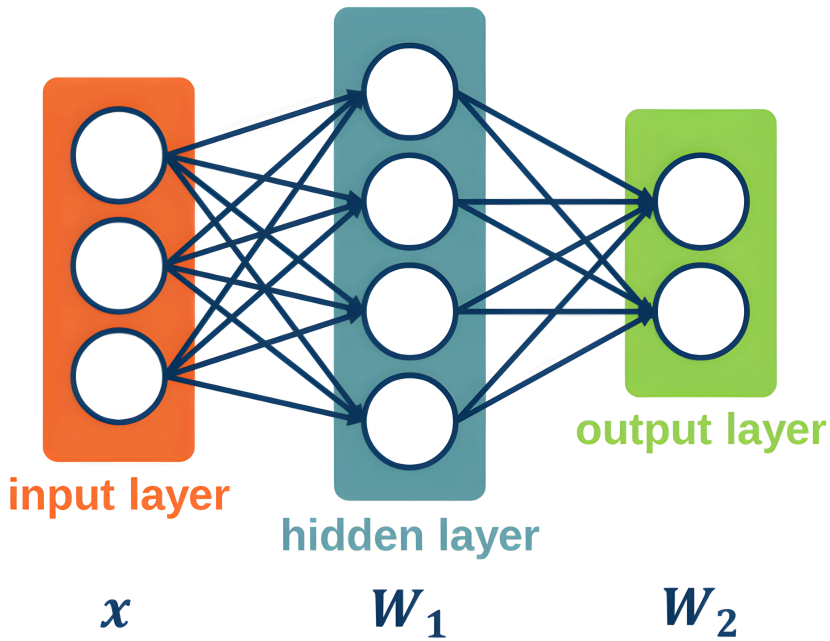
1. Background
2. Quantization
3. Evaluation
4. Outlier Features
5. Discussions

Background


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Layers

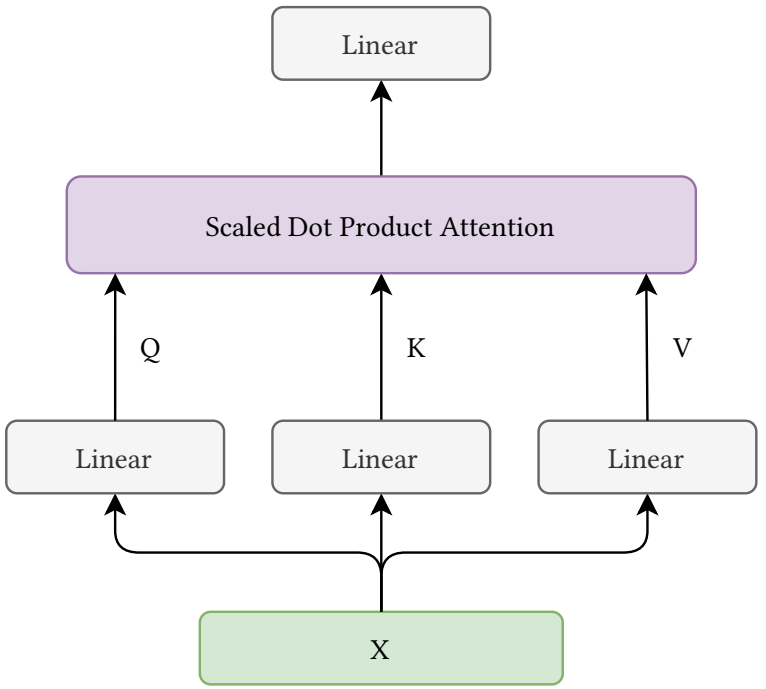
- **MLP**[1]
  - Expansion Layer
  - Contraction Layer
- Attention Block
- Transformer Block



[1]Figure adapted from slides by Zsolt Kira: https://faculty.cc.gatech.edu/~zk15/teaching/AY2024_cs7643_spring/assets/L4_GradientDescent_NNs.pdf

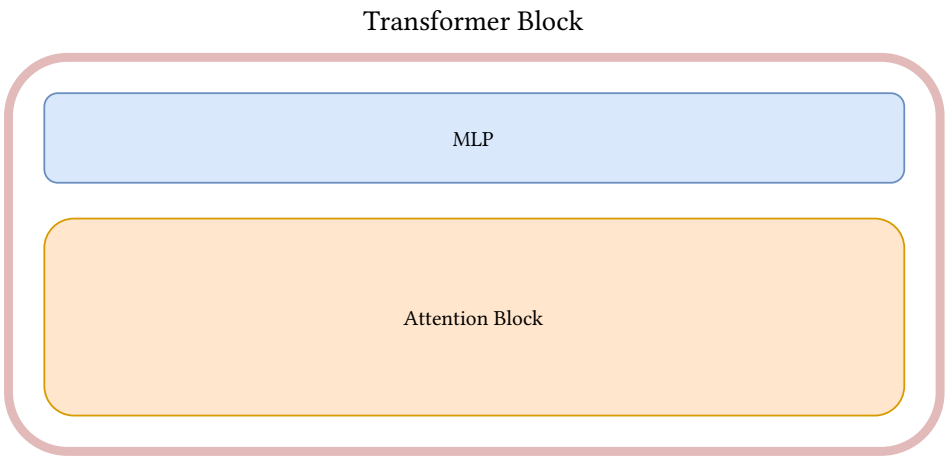
Layers

- MLP
- **Attention Block**
  - 4 Projection (Linear) Layers
  - Key, Value, Query, Output
- Transformer Block



Layers

- MLP
- Attention Block
- **Transformer Block**
  - Attention Block
  - MLP



Scenarios

- Training
  - Forward Pass (value) + Backward Pass (gradient)
  - Forward Pass results need to be stored in memory
- Finetune
  - Similar to training, but sometimes with most of the weights freezed
- Inference
  - Forward Pass only

Scenarios

- Training
- Finetune
- **Inference**

Motivation

- So, what's the problem with the inference?
  - Model is large: OPT-175B/BLOOM
  - Weights need to be loaded into GPU memory
  - 175B parameters (16 bit) -> 8x A100 (80G) -> 💰150000+

Motivation

- So, what's the problem with the inference?
  - Model is large: OPT-175B/BLOOM
  - Weights need to be loaded into GPU memory
  - 175B parameters (16 bit) -> 8x A100 (80G) -> 💰150000+
- Solution: **load "less" weights -> Quantization**

Previous Attempts

- Degrade performance
- Require further tuning
- Only been studied for models with less than 350M parameters

Previous Attempts

- Degrade performance
- Require further tuning
- Only been studied for models with less than 350M parameters

So, how to do **degradation-free quantization** up to **billions of parameters?**

Quantization

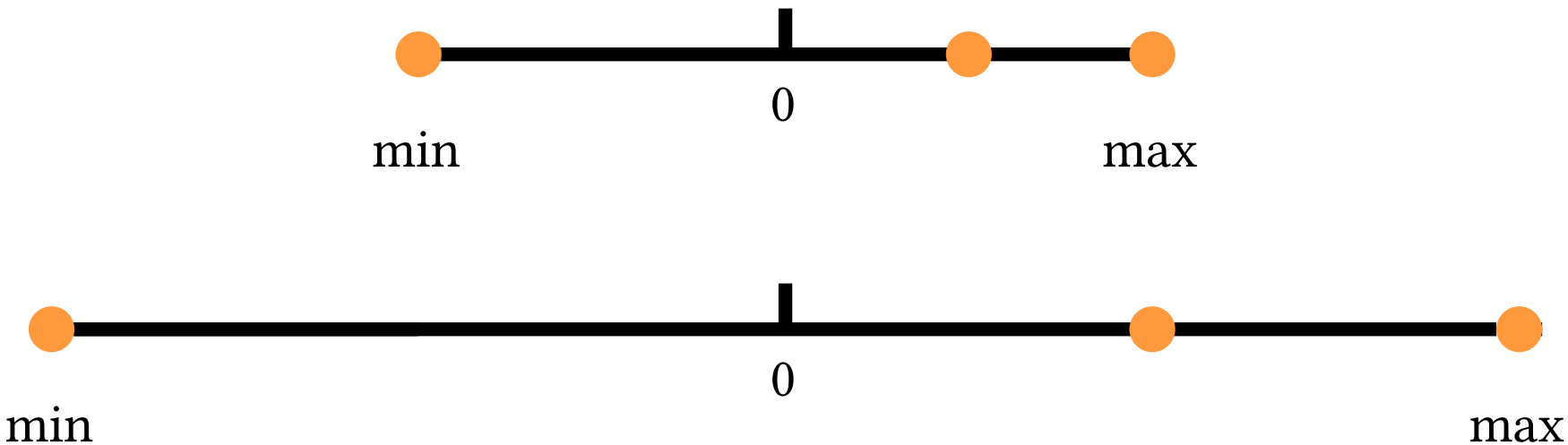
- Quantize fp16 matrices to int8 matrices
- Support matrix multiplication

Absmax

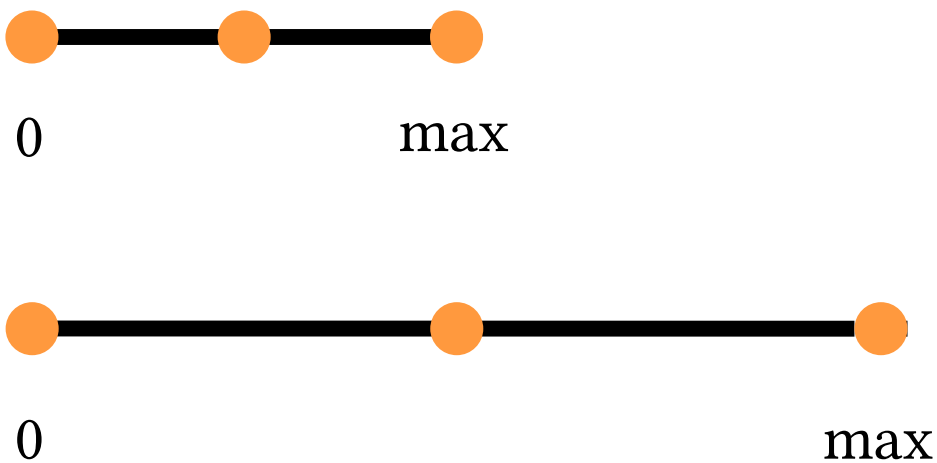
- Absolute Maximum Quantization
- Scale input into $[-127, 127]$ ranges

$$\boldsymbol{X}_{i8} = \left\lfloor \frac{127 \cdot \boldsymbol{X}_{f16}}{\max_{ij}\left(\left|\boldsymbol{X}_{f16}^{ij}\right|\right)} \right\rceil = \left\lfloor s_{x_{f16}} \boldsymbol{X}_{f16} \right\rceil$$

Absmax



Absmax



Zeropoint

- Absmax is bad when the data is asymmetric.
- Shift the input into the range $[-127, 127]$ by **scaling** with the normalized dynamic range nd_x and **shifting** by the zeropoint zp_x

$$nd_{x_{f16}} = \frac{2 \cdot 127}{\max_{ij}\left(\boldsymbol{X}_{f16}^{ij}\right) - \min_{ij}\left(\boldsymbol{X}_{f16}^{ij}\right)}$$

$$zp_{x_{i16}} = \left\lfloor X_{f16} \cdot \min_{ij}\left(\boldsymbol{X}_{f16}^{ij}\right) \right\rceil$$

$$X_{i8} = \left\lfloor nd_{x_{f16}} X_{f16} \right\rceil$$

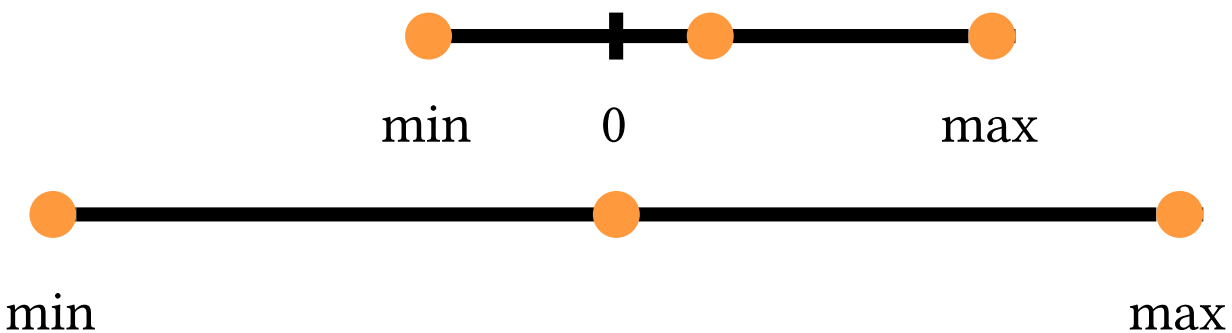
Zeropoint

- However, the formula in the paper is wrong.[2]

[2] Their code for calculating zeropoint is also wrong, but in a different way: see e1f515. A few hours after making the slides, the wrong test was removed from the main branch.

Zeropoint

- However, the formula in the paper is wrong.
- Let's recall our goal:



Zeropoint

$$nd_{x_{f16}} \cdot \min_{ij}\left(\boldsymbol{X}_{f16}^{ij}\right) + zp_{x_{i16}} = -127$$

$$nd_{x_{f16}} \cdot \max_{ij}\left(\boldsymbol{X}_{f16}^{ij}\right) + zp_{x_{i16}} = 127$$

Solving the above equations gives us

$$nd_{x_{f16}} = \frac{2 \cdot 127}{\max_{ij}\left(\boldsymbol{X}_{f16}^{ij}\right) - \min_{ij}\left(\boldsymbol{X}_{f16}^{ij}\right)}$$

$$zp_{x_{i16}} = -\left\lfloor nd_{x_{f16}} \cdot \min_{ij}\left(\boldsymbol{X}_{f16}^{ij}\right) \right\rceil - 127$$

Matrix Multiplication - Naive

Given input $\boldsymbol{X}_{f16} \in R^{s \times h}$ and weights $\boldsymbol{W}_{f16} \in R^{h \times o}$ with sequence dimension s, feature dimension h, and output dimension o,

$$\boldsymbol{X}_{f16}\boldsymbol{W}_{f16} \approx \frac{1}{s_{x_{f16}} s_{w_{f16}}} \cdot \boldsymbol{X}_{i8}\boldsymbol{W}_{i8} \qquad \text{(absmax)}$$

$$\boldsymbol{X}_{f16}\boldsymbol{W}_{f16} \approx \frac{1}{nd_{x_{f16}} nd_{w_{f16}}} \cdot \left(\boldsymbol{X}_{i8} - zp_{x_{i16}}\right) \cdot \left(\boldsymbol{W}_{i8} - zp_{w_{i16}}\right) \qquad \text{(zp)}$$

- zeropoint is not used due to the need to add zp to quantized values[3]

[3] some GPUs/TPUs don't support this

Matrix Multiplication - Naive

$$\boldsymbol{X}_{f16} = \boldsymbol{W}_{f16} = \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \xrightarrow{\text{absmax quantization}} \begin{pmatrix} 64 & 127 \\ -127 & 64 \end{pmatrix}$$

Matrix Multiplication - Naive

$$\boldsymbol{X}_{f16} = \boldsymbol{W}_{f16} = \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \xrightarrow{\text{absmax quantization}} \begin{pmatrix} 64 & 127 \\ -127 & 64 \end{pmatrix}$$

$$\boldsymbol{X}_{i8}\boldsymbol{W}_{i8} = \begin{pmatrix} -12033 & 16256 \\ -16256 & -12033 \end{pmatrix} \xrightarrow{\text{dequantization}} \begin{pmatrix} -2.98 & 4.03 \\ -4.03 & -2.98 \end{pmatrix}$$

Matrix Multiplication - Naive

$$\boldsymbol{X}_{f16} = \boldsymbol{W}_{f16} = \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \xrightarrow{\text{absmax quantization}} \begin{pmatrix} 64 & 127 \\ -127 & 64 \end{pmatrix}$$

$$\boldsymbol{X}_{i8}\boldsymbol{W}_{i8} = \begin{pmatrix} -12033 & 16256 \\ -16256 & -12033 \end{pmatrix} \xrightarrow{\text{dequantization}} \begin{pmatrix} -2.98 & 4.03 \\ -4.03 & -2.98 \end{pmatrix}$$

$$\boldsymbol{X}_{f16}\boldsymbol{W}_{f16} = \begin{pmatrix} -3 & 4 \\ -4 & -3 \end{pmatrix}$$

Matrix Multiplication - Naive

Let's consider a different input matrix

$$\boldsymbol{X}_{f16} = \begin{pmatrix} 3 & 508 \\ 1 & 1 \end{pmatrix}$$

Matrix Multiplication - Naive

Let's consider a different input matrix

$$\boldsymbol{X}_{f16} = \begin{pmatrix} 3 & 508 \\ 1 & 1 \end{pmatrix}$$

After quantization, we have

$$\boldsymbol{X}_{f16} = \begin{pmatrix} 1 & 127 \\ 0 & 0 \end{pmatrix}$$

Matrix Multiplication - Rowwise

- Quantize each row of $\boldsymbol{X}_{f16}$ independently

Matrix Multiplication - Rowwise

- Quantize each row of $\boldsymbol{X}_{f16}$ independently

$$\boldsymbol{X}_{f16} = \begin{pmatrix} 3 & 508 \\ 1 & 1 \end{pmatrix} \xrightarrow{\text{rowwise quantization}} \begin{pmatrix} 1 & 127 \\ 127 & 127 \end{pmatrix}, \quad S_{x_{f16}} = \begin{pmatrix} 0.25 \\ 127 \end{pmatrix}$$

Matrix Multiplication - Rowwise

- Quantize each row of $\boldsymbol{X}_{f16}$ independently

$$\boldsymbol{X}_{f16} = \begin{pmatrix} 3 & 508 \\ 1 & 1 \end{pmatrix} \xrightarrow{\text{rowwise quantization}} \begin{pmatrix} 1 & 127 \\ 127 & 127 \end{pmatrix}, \quad S_{x_{f16}} = \begin{pmatrix} 0.25 \\ 127 \end{pmatrix}$$

$$\boldsymbol{W}_{f16} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \xrightarrow{\text{absmax quantization}} \begin{pmatrix} 64 \\ 127 \end{pmatrix}, \quad S_{w_{f16}} = 63.5$$

Matrix Multiplication - Rowwise

- Quantize each row of $\boldsymbol{X}_{f16}$ independently

$$\boldsymbol{X}_{f16} = \begin{pmatrix} 3 & 508 \\ 1 & 1 \end{pmatrix} \xrightarrow{\text{rowwise quantization}} \begin{pmatrix} 1 & 127 \\ 127 & 127 \end{pmatrix}, \quad S_{x_{f16}} = \begin{pmatrix} 0.25 \\ 127 \end{pmatrix}$$

$$\boldsymbol{W}_{f16} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \xrightarrow{\text{absmax quantization}} \begin{pmatrix} 64 \\ 127 \end{pmatrix}, \quad S_{w_{f16}} = 63.5$$

$$\boldsymbol{X}_{f16}\boldsymbol{W}_{f16} = \begin{pmatrix} 1019 \\ 3 \end{pmatrix} \quad \text{and} \quad \boldsymbol{X}_{i8}\boldsymbol{W}_{i8} = \begin{pmatrix} 16193 \\ 24257 \end{pmatrix} \xrightarrow[\text{div by } S_{x_{f16}} \times S_{w_{f16}}]{\text{dequantization}} \begin{pmatrix} 1020.03 \\ 3.01 \end{pmatrix}$$

Matrix Multiplication - Rowwise

Let's consider a different weight matrix

$$\boldsymbol{W}_{f16} = \begin{pmatrix} 3 & 508 \\ 1 & 1 \end{pmatrix}$$

Matrix Multiplication - Rowwise

Let's consider a different weight matrix

$$\boldsymbol{W}_{f16} = \begin{pmatrix} 3 & 508 \\ 1 & 1 \end{pmatrix}$$

After quantization, we have

$$\begin{pmatrix} 1 & 127 \\ 0 & 0 \end{pmatrix}$$

Matrix Multiplication - Vectorwise

Observations: matmul are a sequence of independent inner products

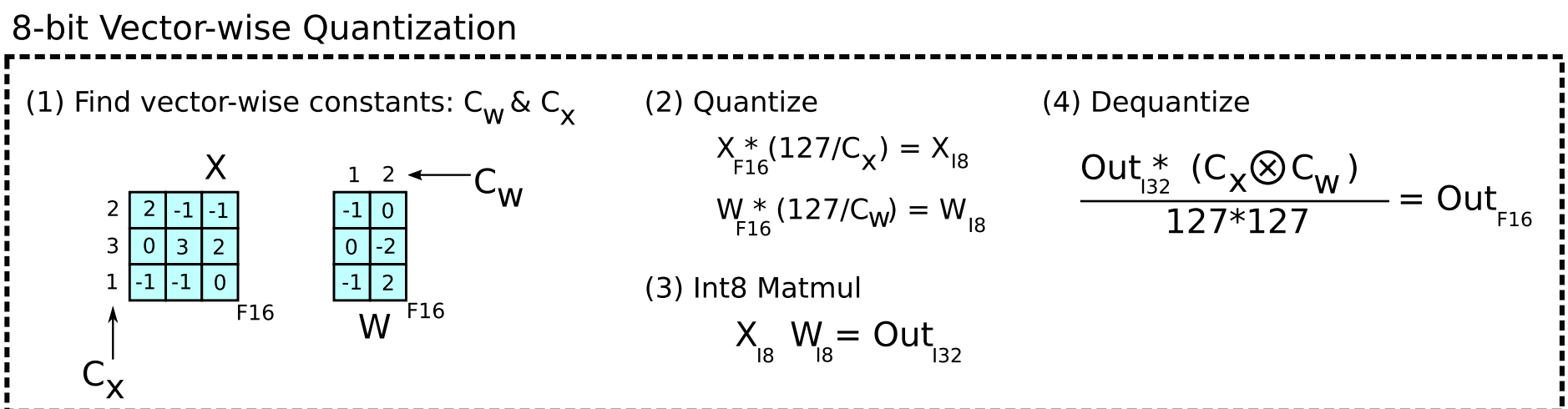
- $XW_{ab} = \sum_h X_{ah} W_{hb} = \langle X_a, W_b^T \rangle$

Matrix Multiplication - Vectorwise

Observations: matmul are a sequence of independent inner products

- $XW_{ab} = \sum_h X_{ah} W_{hb} = \langle X_a, W_b^T \rangle$

Quantize each *column* of $\boldsymbol{W}$ independently



Matrix Multiplication - Vectorwise

$$\mathrm{Out}_{\mathrm{I32}_{11}} = \left\langle X_{\mathrm{I8}_1}, W^T_{\mathrm{I8}_1} \right\rangle \approx C_{X_1} C_{W_1} \left(X_{\mathrm{F16}_1} W_{\mathrm{F16}_1} \right)$$

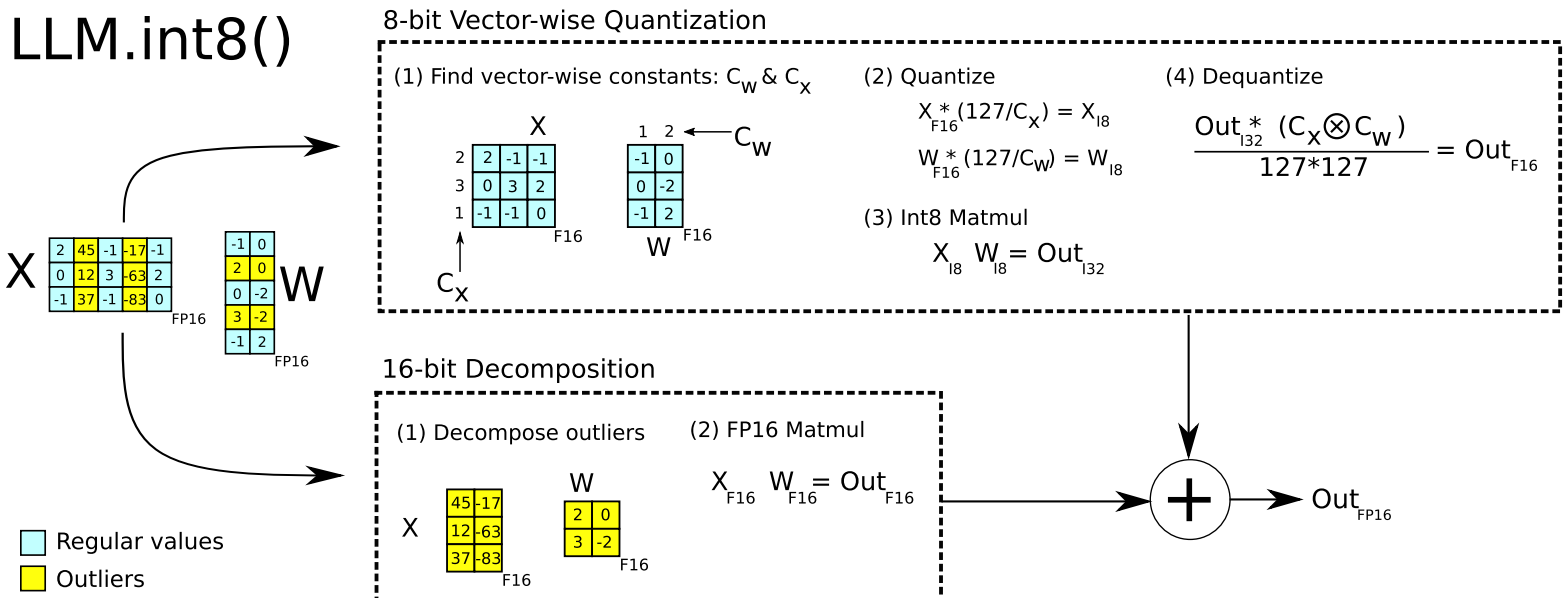
Matrix Multiplication - Vectorwise

$$\mathrm{Out}_{\mathrm{I32}_{11}} = \left\langle X_{\mathrm{I8}_1}, W^T_{\mathrm{I8}_1} \right\rangle \approx C_{X_1} C_{W_1} \left(X_{\mathrm{F16}_1} W_{\mathrm{F16}_1} \right)$$

$$C_X C_W = \begin{pmatrix} 2 & 4 \\ 3 & 6 \\ 1 & 2 \end{pmatrix}$$

- $C_{X_1} C_{W_1}$ is the topleft element in the outer product

Mixed-precision Decomposition



- Consistent outlier values in columns of X + Do separate matmul in fp16
- Luckily, they are sparse (only 0.1%). Is LLM.int8() actually int8 🤔?

LLM.int8 in Practice

- Load the weight from fp16/32 checkpoints
- Quantize it to int8 and send to GPUs
- When fp16 matmul is needed, weight matrix is dequantized to fp16

Evaluation

- Question: how well `LLM.int8()` performs as the model size **scales**.
  - Models: 125M - 125B
- Two Settings: Language Modelling + End Tasks
- Language Modelling
  - Dataset: C4 corpus
  - Metric: Perplexity (robust, sensitive to quantization degradation)
- End Tasks
  - Dataset: EleutherAI language model evaluation harness
  - Metric: Zero-shot Accuracy

Evaluation Results

Parameters	125M	1.3B	2.7B	6.7B	13B
32-bit Float	25.65	15.91	14.43	13.30	12.45
Int8 absmax	87.76	16.55	15.11	14.59	19.08
Int8 zeropoint	56.66	16.24	14.76	13.49	13.94
Int8 absmax row-wise	30.93	17.08	15.24	14.13	16.49
Int8 absmax vectorwise	35.84	16.82	14.98	14.13	16.48
Int8 zeropoint vectorwise	25.72	15.94	14.36	13.38	13.47
Int8 absmax rowwise + decomp	30.76	16.19	14.65	13.25	12.46
Absmax LLM.int8() vectorwise + decomp	25.83	15.93	14.44	13.24	12.45
Zp LLM.int8() vectorwise + decomp	25.69	15.92	14.43	13.24	12.45

Table 1: C4 validation perplexities of quantization methods for different transformer sizes.

Evaluation Results

Parameters	125M	1.3B	2.7B	6.7B	13B
32-bit Float	25.65	15.91	14.43	13.30	12.45
Int8 absmax	87.76	16.55	15.11	14.59	19.08
Int8 zeropoint	56.66	16.24	14.76	13.49	13.94
Int8 absmax row-wise	30.93	17.08	15.24	14.13	16.49
Int8 absmax vectorwise	35.84	16.82	14.98	14.13	16.48
Int8 zeropoint vectorwise	25.72	15.94	14.36	13.38	13.47
Int8 absmax rowwise decomp	30.76	16.19	14.65	13.25	12.46
Absmax LLM.int8() vectorwise decomp	25.83	15.93	14.44	**13.24**	**12.45**
Zp LLM.int8() vectorwise decomp	**25.69**	**15.92**	**14.43**	**13.24**	**12.45**

- Performance degrades as model scales + LLM.int8 has comparable perf

Evaluation Results

Parameters	125M	1.3B	2.7B	6.7B	13B
32-bit Float	25.65	15.91	14.43	13.30	12.45
Int8 absmax	87.76	16.55	15.11	14.59	19.08
Int8 zeropoint	56.66	16.24	14.76	13.49	13.94
Int8 absmax row-wise	30.93	17.08	15.24	14.13	16.49
Int8 absmax vectorwise	35.84	16.82	14.98	14.13	16.48
Int8 zeropoint vectorwise	25.72	15.94	14.36	13.38	13.47
Int8 absmax rowwise + decomp	30.76	16.19	14.65	13.25	12.46
Absmax LLM.int8() vectorwise + decomp	25.83	15.93	14.44	13.24	12.45
Zp LLM.int8() vectorwise + decomp	25.69	15.92	14.43	13.24	12.45

- With decomposition, zeropoint is no longer advantageous

Outlier Features

- Given a hidden state $X \in R^{s \times h}$ where s is the sequence/token dimension and h the hidden/feature dimension, a **feature** is a particular dimension h_i.

Outlier Features

- Given a hidden state $X \in R^{s \times h}$ where s is the sequence/token dimension and h the hidden/feature dimension, a **feature** is a particular dimension h_i.

How to determine if a feature is an outlier?

Outlier Features

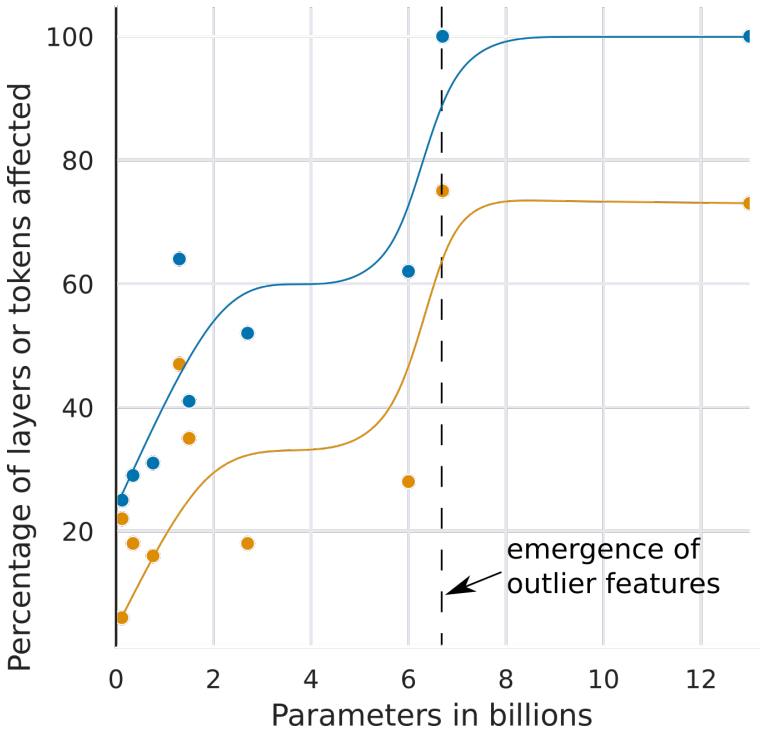
- 4 Projection layers in Attention + Expansion layer in MLP
- Outliers
  - Treat features ≥ 6 as outliers $\Rightarrow$ no more perplexity degradation
  - *Systematic* in large models: either in most layers or not at all
  - *Probabilistic* in small models: sometimes in some layers
- Find thresholds to limit detection to a single outlier in the smallest model with 125M parameters

Outlier Features - Threshold

- The feature value is at least 6.0
- It occurs in the same h_i in at least 25% of layers
- It appears in at least 6% of the sequence dimensions

Understanding Outlier Features

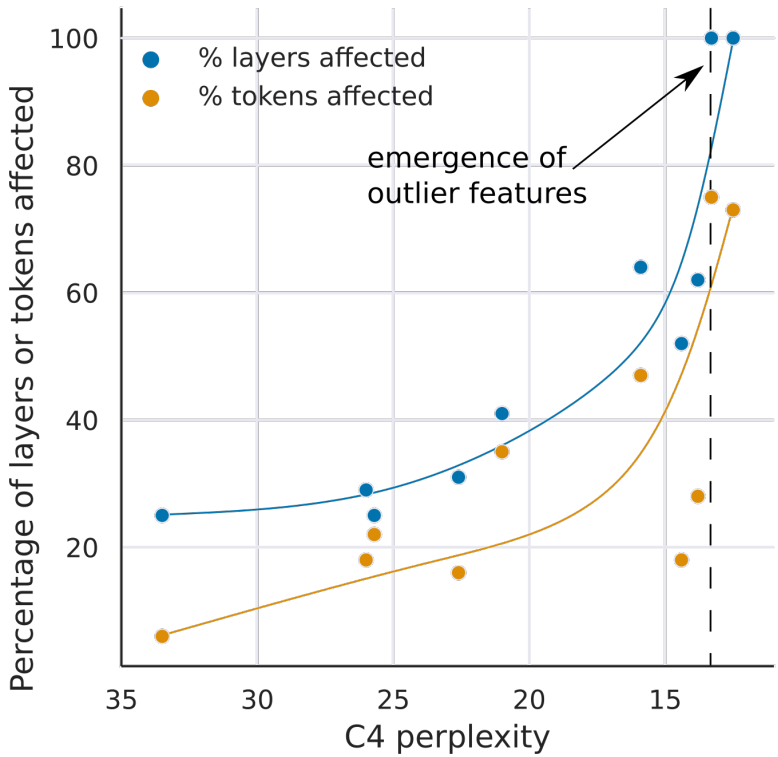
- **Emergence**[4]
  - Sudden increase of presence in feature (blue) and sequence dimensions (orange) as model size increases
- Systematic after phase shift
- Critical



[4]The following studies are done in a series of models up to 13B parameters.

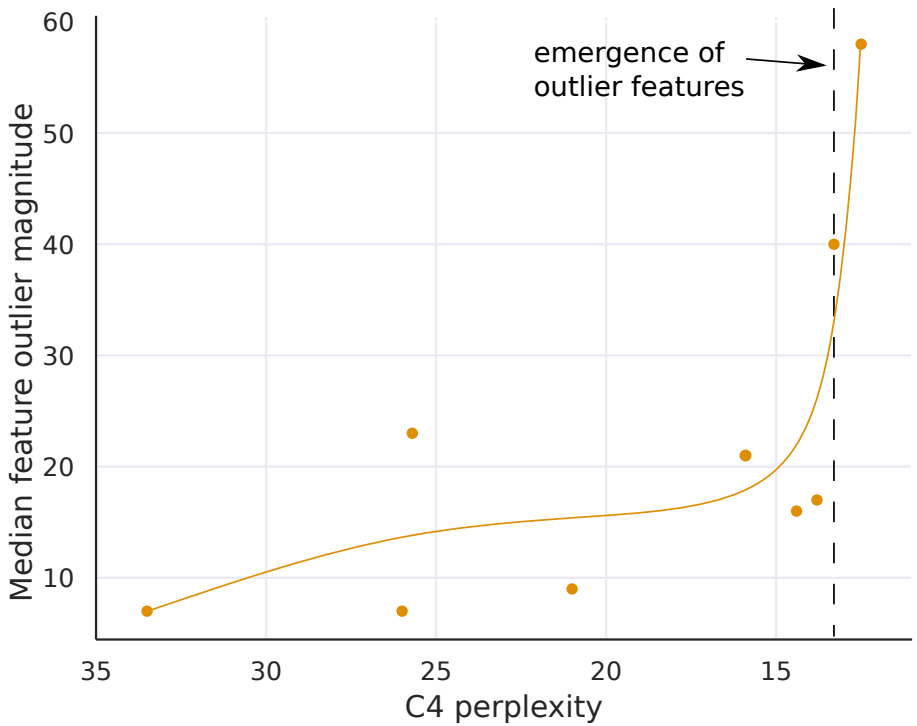
Understanding Outlier Features

- **Emergence**
  - Sudden increase of presence in feature (blue) and sequence dimensions (orange) as model size increases
  - Nothing sudden in terms of perplexity
- Systematic after phase shift
- Critical



Understanding Outlier Features

- **Emergence**
  - Sudden increase of presence in feature (blue) and sequence dimensions (orange) as model size increases
  - Nothing sudden in terms of perplexity
  - Median rapidly increases
- Systematic after phase shift
- Critical



Understanding Outlier Features

- Emergence
- **Systematic** after phase shift
  - For a 6.7B transformer with a sequence length of 2048,
    - 150k outlier features per sequence
    - Concentrated in only *6* different hidden dimensions
- Critical

Understanding Outlier Features

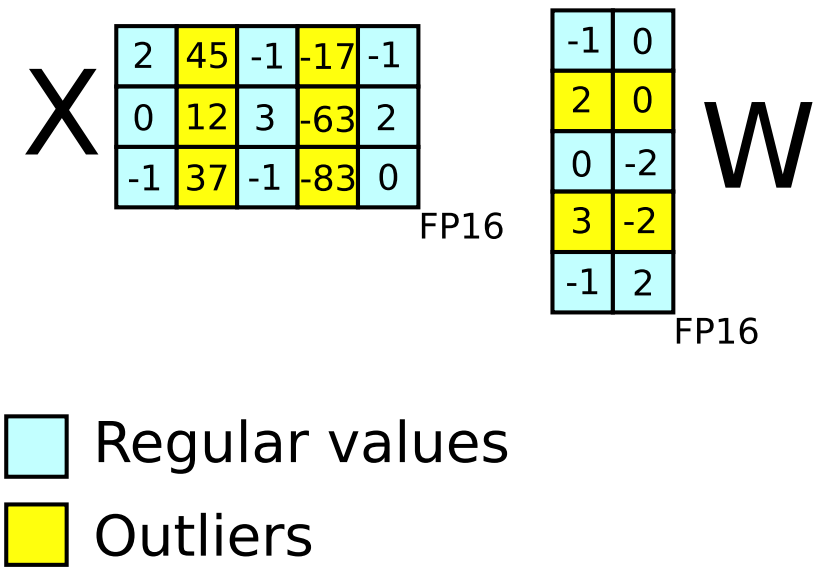
- Emergence
- Systematic after phase shift
- **Critical**
  - Even if there are only 7 outliers, after removing them,
    - the mean top-1 softmax probability is reduced from 40% to 20%
    - the perplexity increases by 600-1000%
  - Remove 7 random features,
    - the top-1 probability decreases only between 0.02-0.3%
    - the perplexity increases by 0.1%

Outlier Features in Practice

- Any problems in outlier detections? 👀

Outlier Features in Practice

- Outlier features are context-dependent[5]
  - Different outliers can show at different positions for different input
- In practice, outlier detection works on weight matrix instead[6]



[5]The author uses this word in his blog: https://timdettmers.com/2022/08/17/llm-int8-and-emergent-features/

[6]See their code here: https://github.com/bitsandbytes-foundation/bitsandbytes/blob/8b6fe9eef1a2f93af701e020b9a757e43b18a42f/bitsandbytes/utils.py#L8-L103

Discussions

- Limitations
  - Focus on int8 only (no fp8)
  - Only study models up to 175B params
- Int8 training and finetuning performance are not ideal

- Small-scale language models is close to baseline performance
- Performance degrades when int8 is used in attention projection layers even with decomposition

Params	Is 8-bit MLP	Out Proj	Attn	Decomp	PPL
209M				0%	16.74
209M	✓			0%	16.77
209M	✓	✓		0%	16.83
209M	✓	✓		2%	16.78
209M	✓	✓		5%	16.77
209M	✓	✓		10%	16.80
209M	✓	✓	✓	2%	24.33
209M	✓	✓	✓	5%	20.00
209M	✓	✓	✓	10%	19.00
1.1B				0%	9.99
1.1B	✓			0%	9.93
1.1B	✓	✓		0%	10.52
1.1B	✓	✓		1%	10.41

Discussions - Speedup

GPT-3 Size	Small	Medium	Large	XL	2.7B	6.7B	13B	175B
Model dimension	768	1024	1536	2048	2560	4096	5140	12288
FP16-bit baseline	1.00x	1.00x	1.00x	1.00x	1.00x	1.00x	1.00x	1.00x
Int8 no overhead	0.99x	1.08x	1.43x	1.61x	1.63x	1.67x	2.13x	2.29x
Vectorwise	0.43x	0.49x	0.74x	0.91x	0.94x	**1.18x**	**1.59x**	**2.00x**
LLM.int8() vectorwise decomp	0.14x	0.20x	0.36x	0.51x	0.64x	0.86x	**1.22x**	**1.81x**

- Ideal Speedup: the 8-bit without overhead assumes no quantization or dequantization is performed.
- Speed up only for models of large sizes. Why?

Thank you