Efficient Inference: A Deeper Look into Quantization

May 12, 2025

Recap: Representing numbers with float32

Sign
(1 bit)

Exponent
(8 bits)

Fraction
(23 bits)

Key facts

8 bit biased exponent $\rightarrow \approx [10^{-38}, 10^{38}]$

24 bit fraction $\rightarrow$ **7 digit precision**

Recap: bfloat16

Sign
(1 bit)

Exponent
(8 bits)

Fraction
(7 bits)

Key facts

8 bit biased exponent $\rightarrow \approx [10^{-38}, 10^{38}]$ **Same as float32**

7 bit fraction $\rightarrow$ **3 digit precision**

Recap: int8

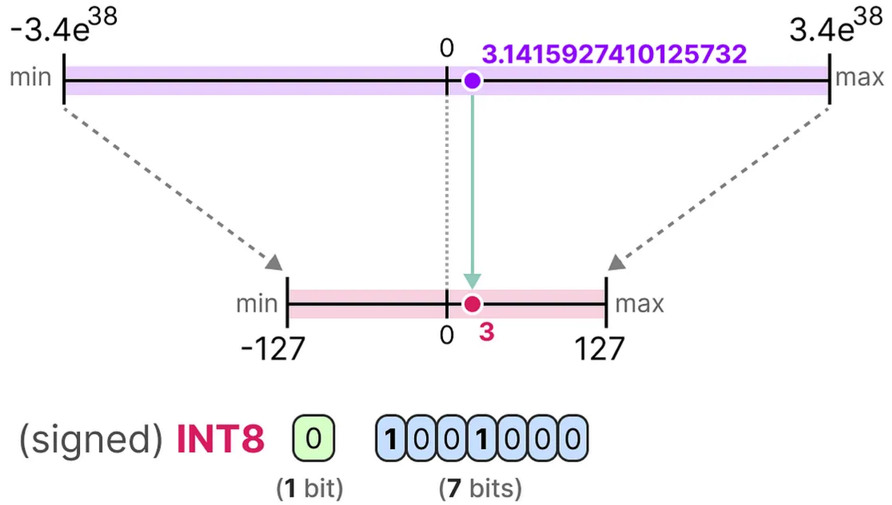
- Can represent 2^8 -1 = 255 values
- Range from [-128, 127]

Key facts

7 bits $\rightarrow [-128, 127]$

255 values only

Recap: Quantizing weights to int8



- **Quantization:** $\mathbf{W}_q = \text{round}\left(\gamma \cdot \mathbf{W}\right)$ with $\gamma = \frac{2^{N-1}}{\max(|\mathbf{W}|)}$ and N=8
- **Dequantization:** $\hat{\mathbf{W}} = \frac{\mathbf{W}_q}{\gamma}$

[Grootendorst]

This lecture

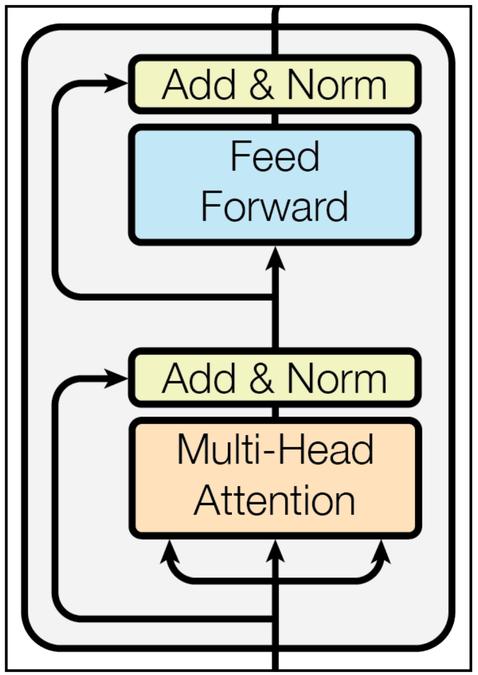
Explore various quantization techniques
and learn about their pros and cons

A closer look at model parameters

- Most computations within the model consist of

$$\mathbf{Y = X\,W + B}$$

- Let us ignore the bias (B) computation
- Consist of much smaller portion of the computation
  - $O(D^2)$ operations in **XW**
  - $O(D)$ operations in **B**
- Biases often not quantized



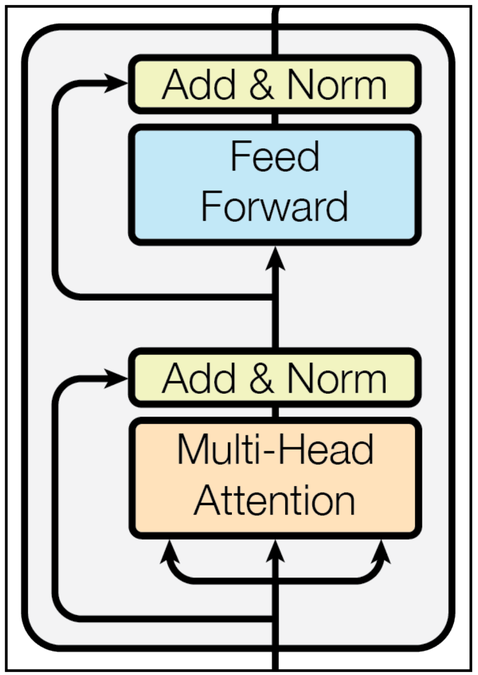
Vaswani et al.

A closer look at model parameters

- Most computations within the model consist of

$$\mathbf{Y = X\,W}$$

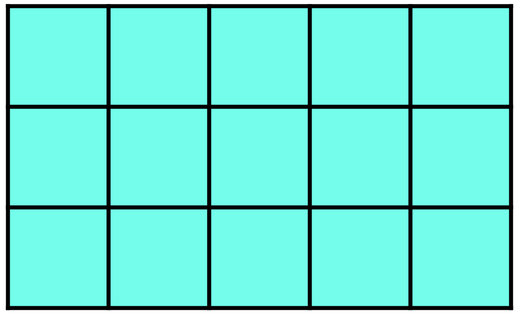
- $\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$ where B is the batch size
- Number of tokens could be another tensor dimensions
  - Will assume it to be 1 unless specified otherwise
- $\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$
- The computation XW projects an input from D_{in} to D_{out} dimensions



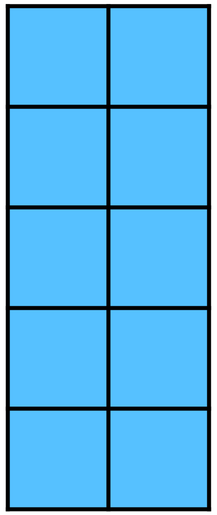
Vaswani et al.

A recap of matrix multiplication

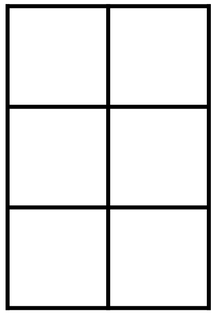
$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$



$\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$



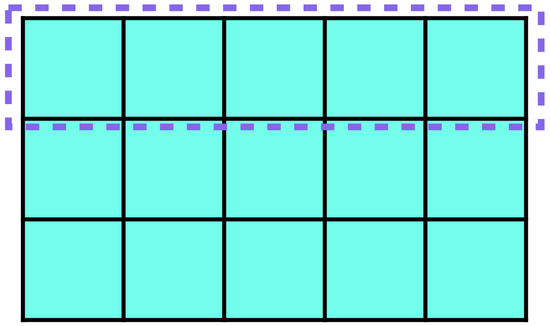
$\mathbf{Y} \in \mathbb{R}^{B \times D_{out}}$



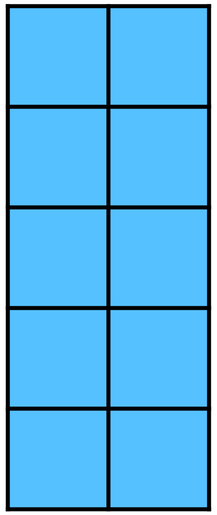
A recap of matrix multiplication

$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$

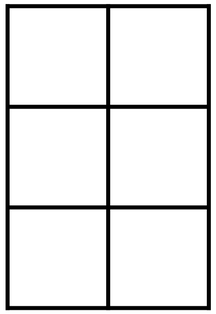
$\mathbf{X}_{[0,:]}$



$\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$



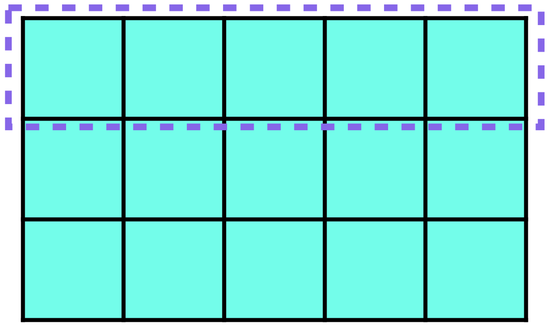
$\mathbf{Y} \in \mathbb{R}^{B \times D_{out}}$



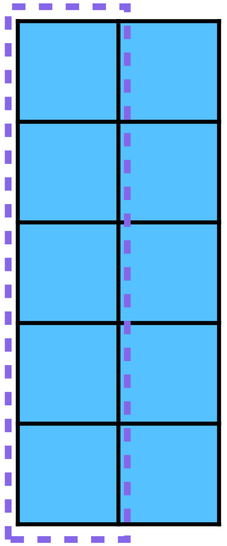
A recap of matrix multiplication

$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$

$\mathbf{X}_{[0,:]}$

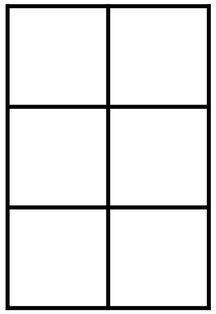


$\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$

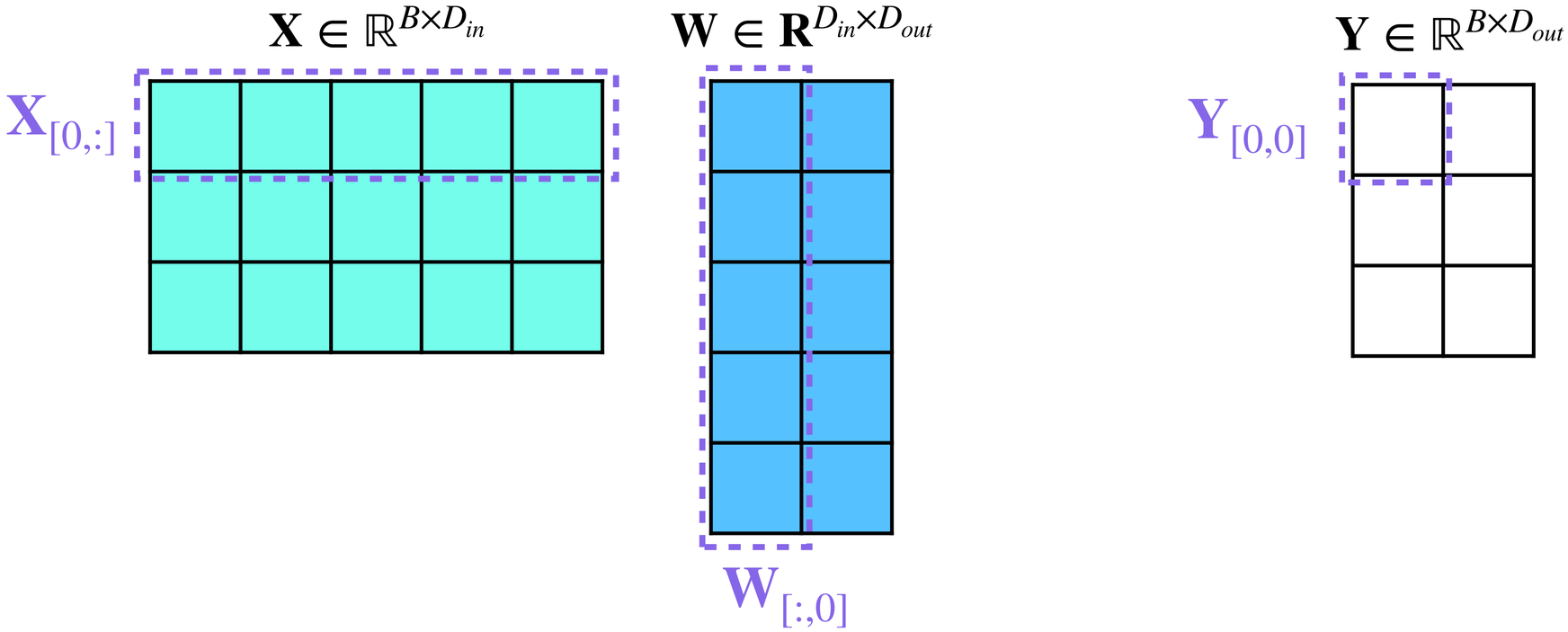


$\mathbf{W}_{[:,0]}$

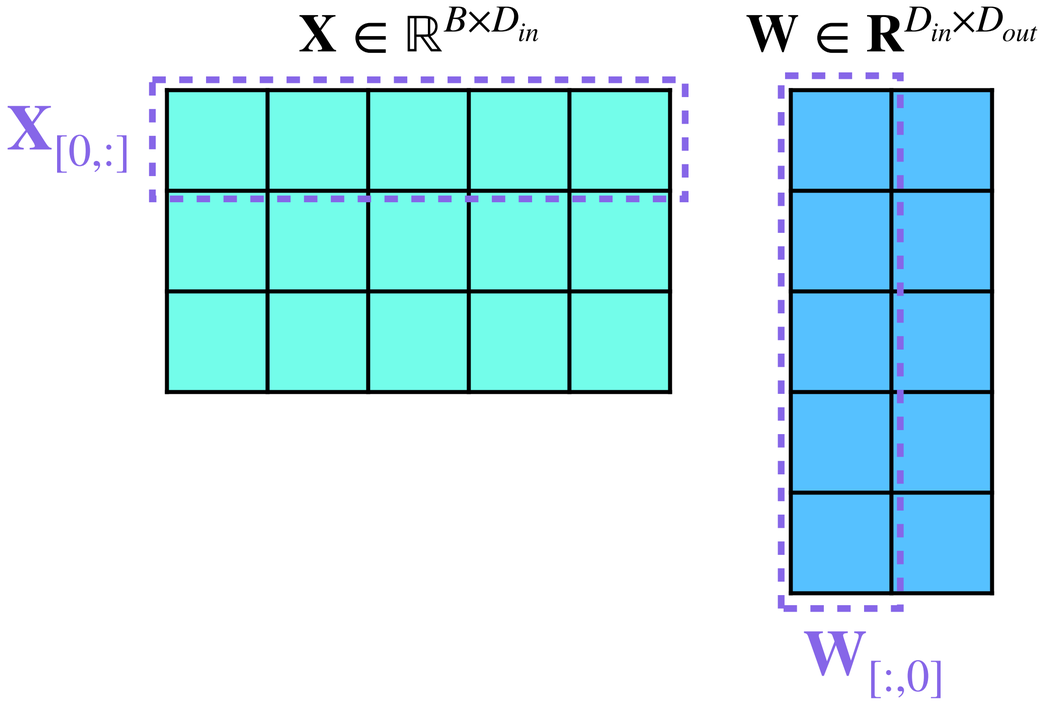
$\mathbf{Y} \in \mathbb{R}^{B \times D_{out}}$

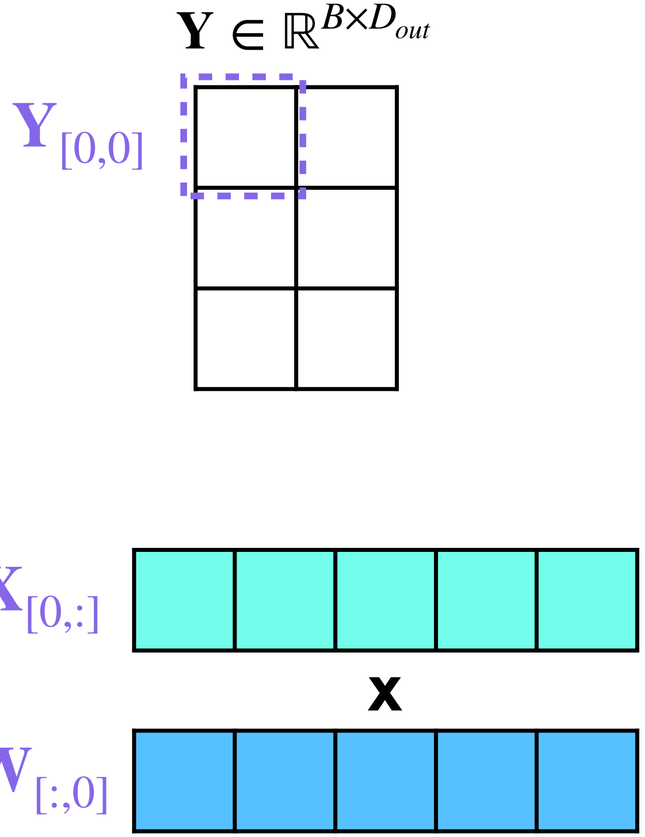


A recap of matrix multiplication

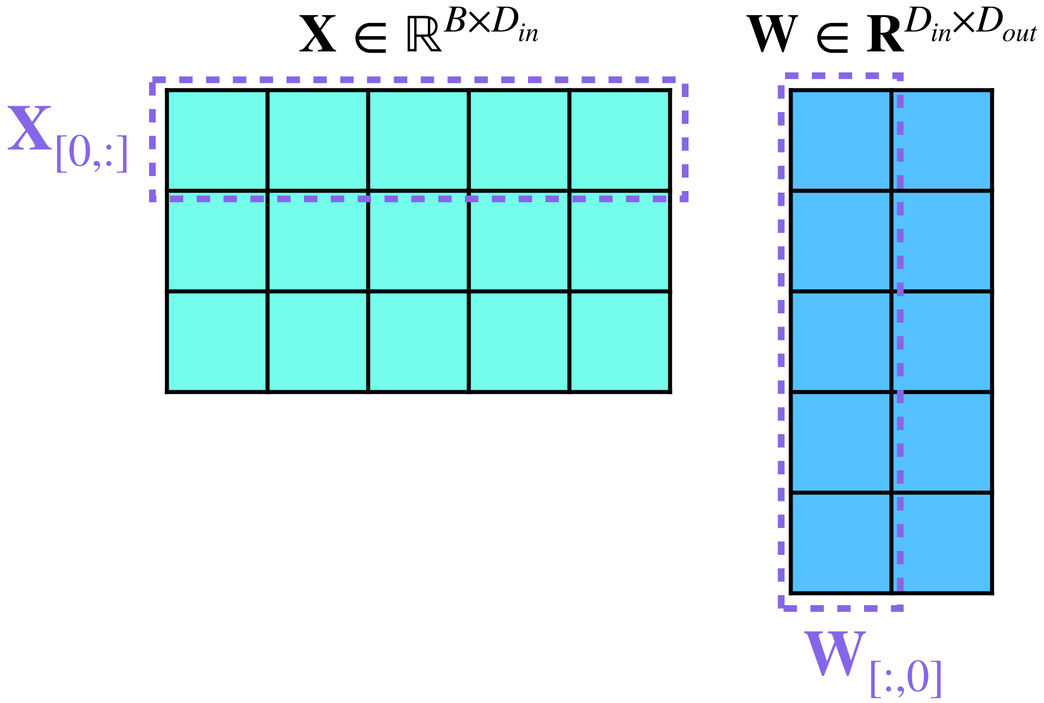


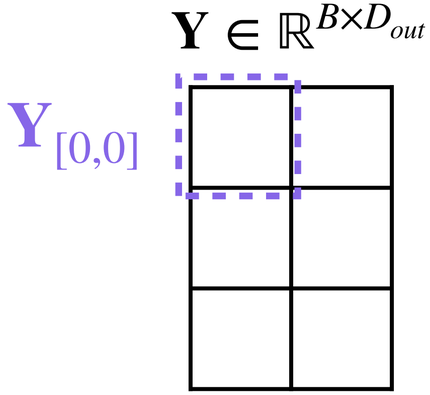
A recap of matrix multiplication

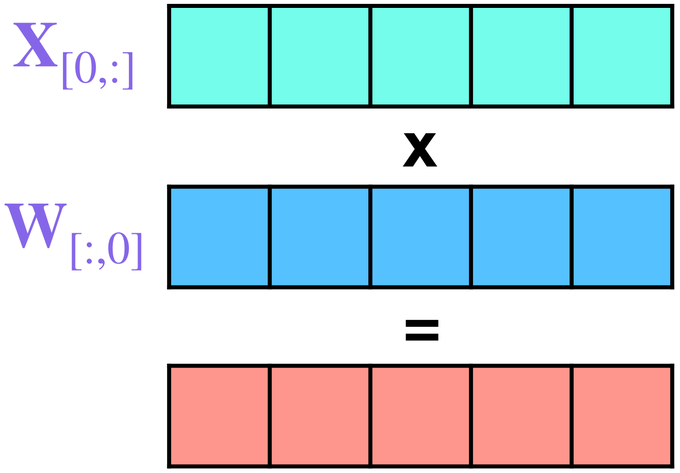




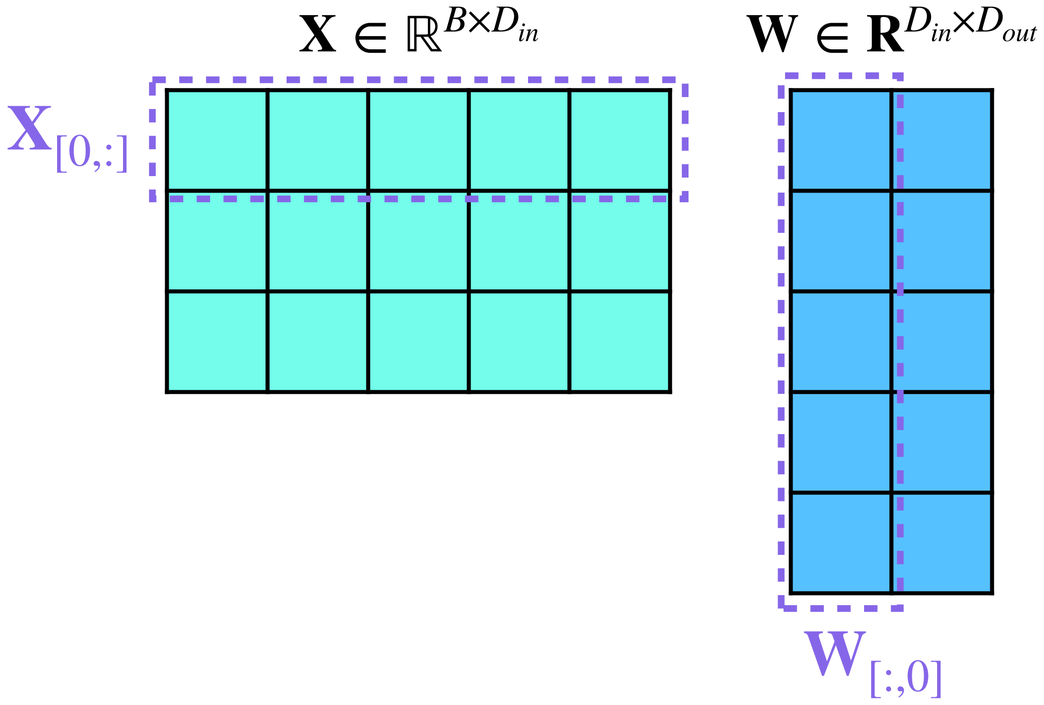
A recap of matrix multiplication

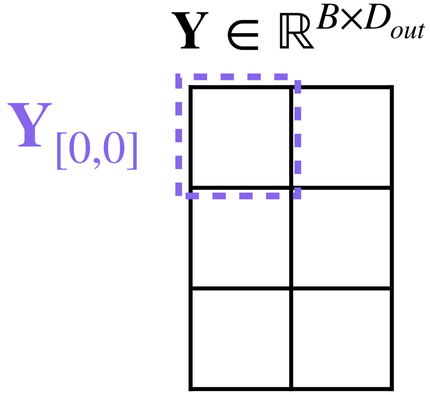


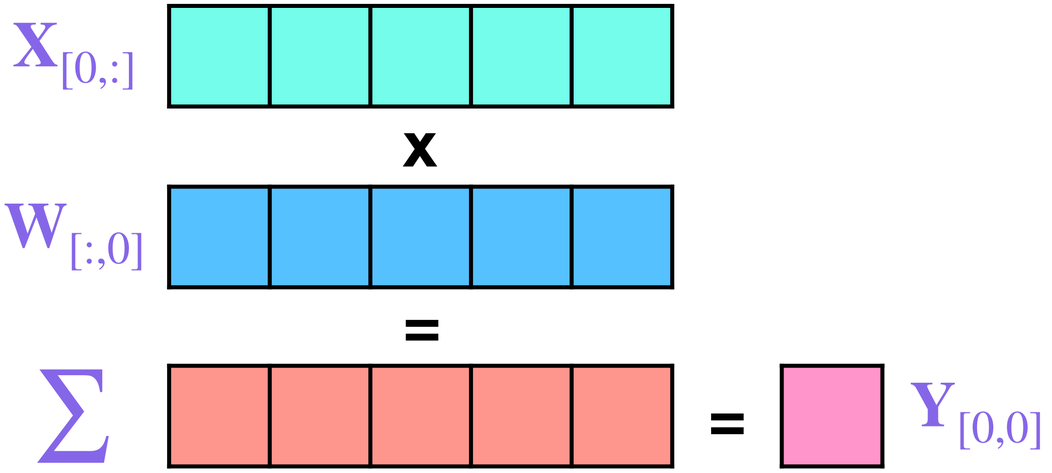




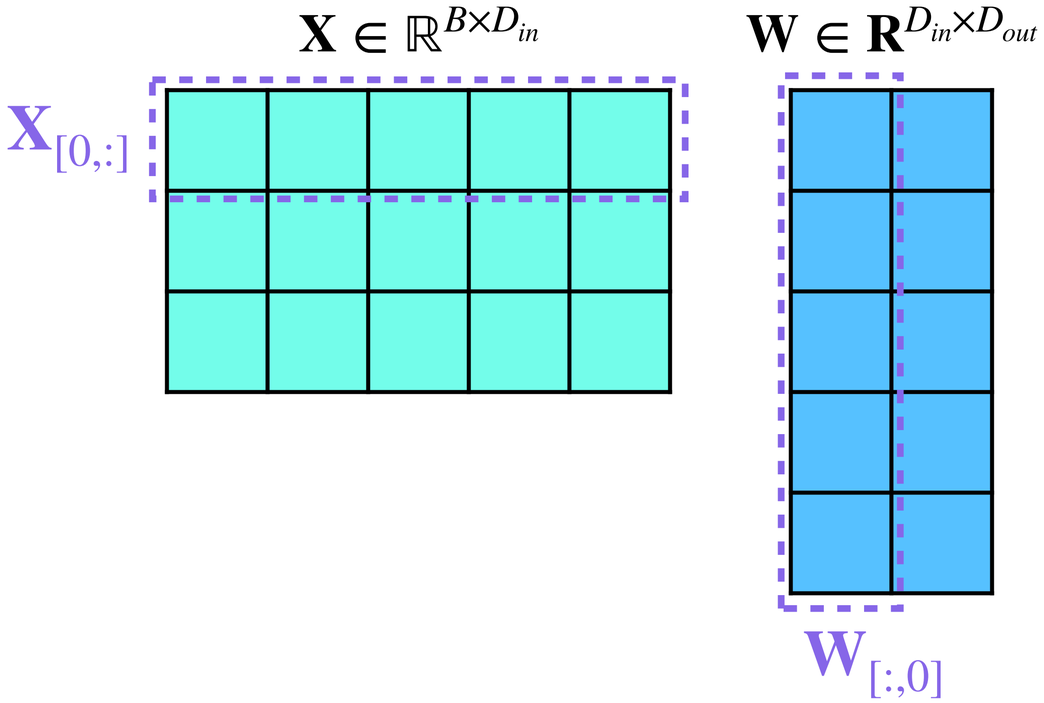
A recap of matrix multiplication

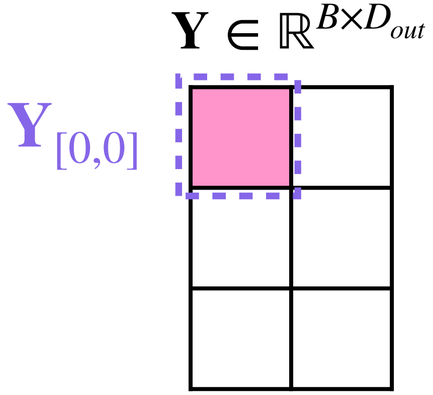


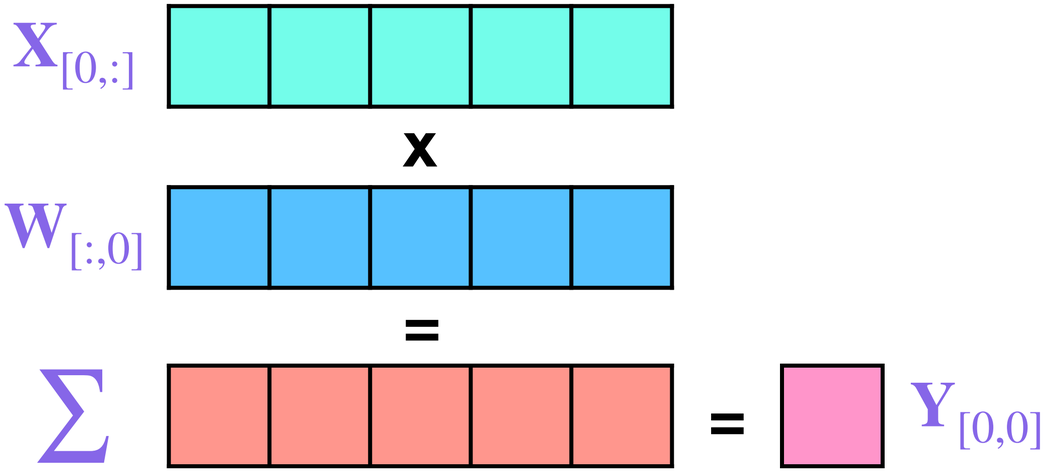




A recap of matrix multiplication



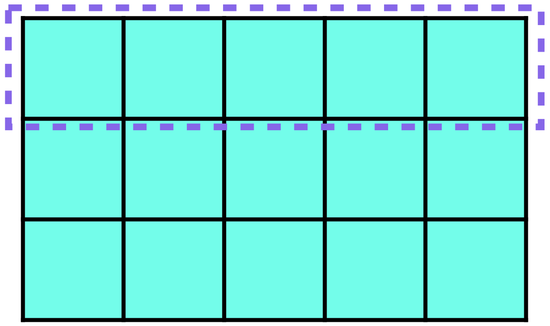




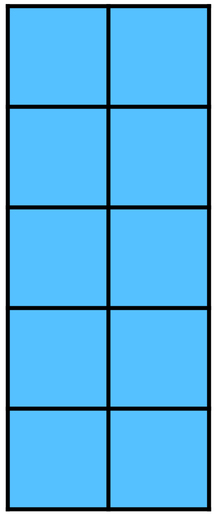
A recap of matrix multiplication

$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$

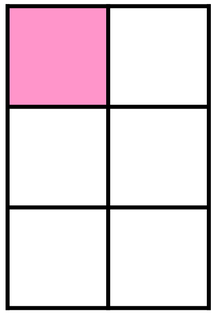
$\mathbf{X}_{[0,:]}$



$\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$



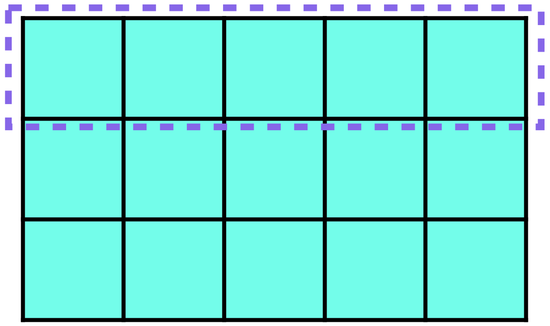
$\mathbf{Y} \in \mathbb{R}^{B \times D_{out}}$



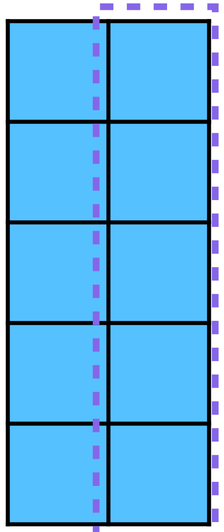
A recap of matrix multiplication

$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$

$\mathbf{X}_{[0,:]}$

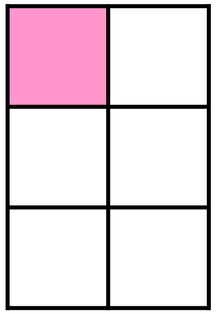


$\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$

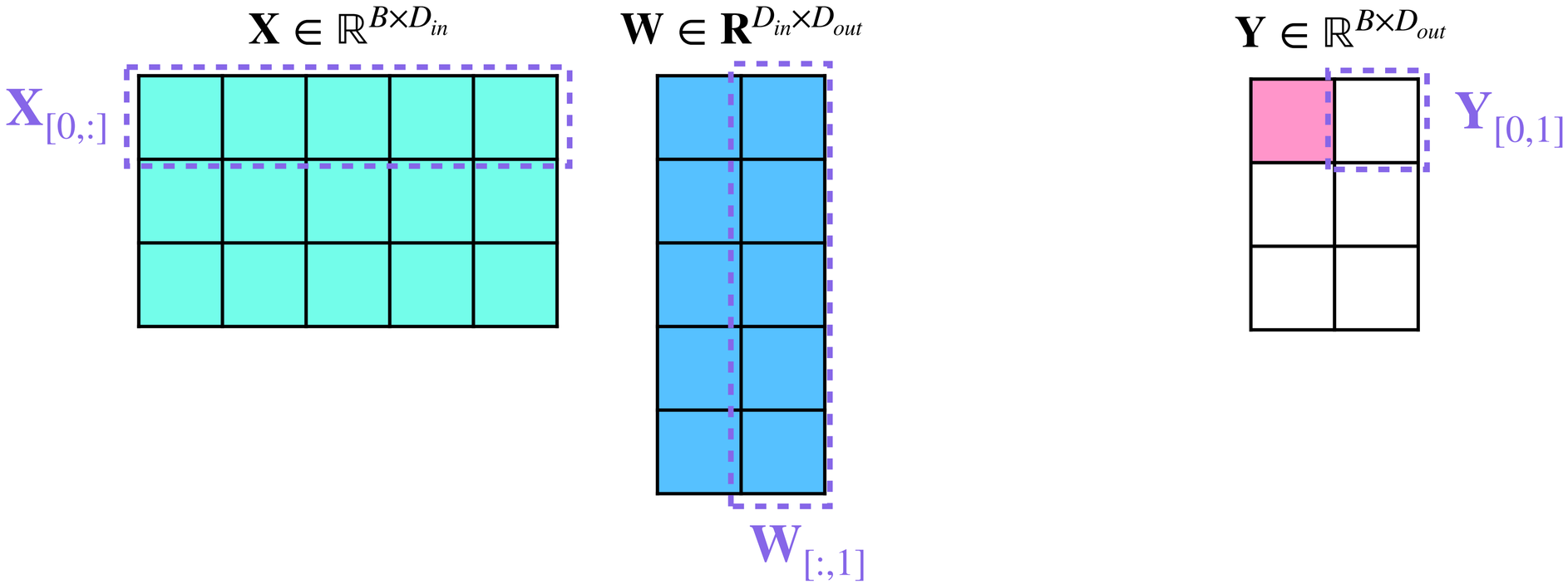


$\mathbf{W}_{[:,1]}$

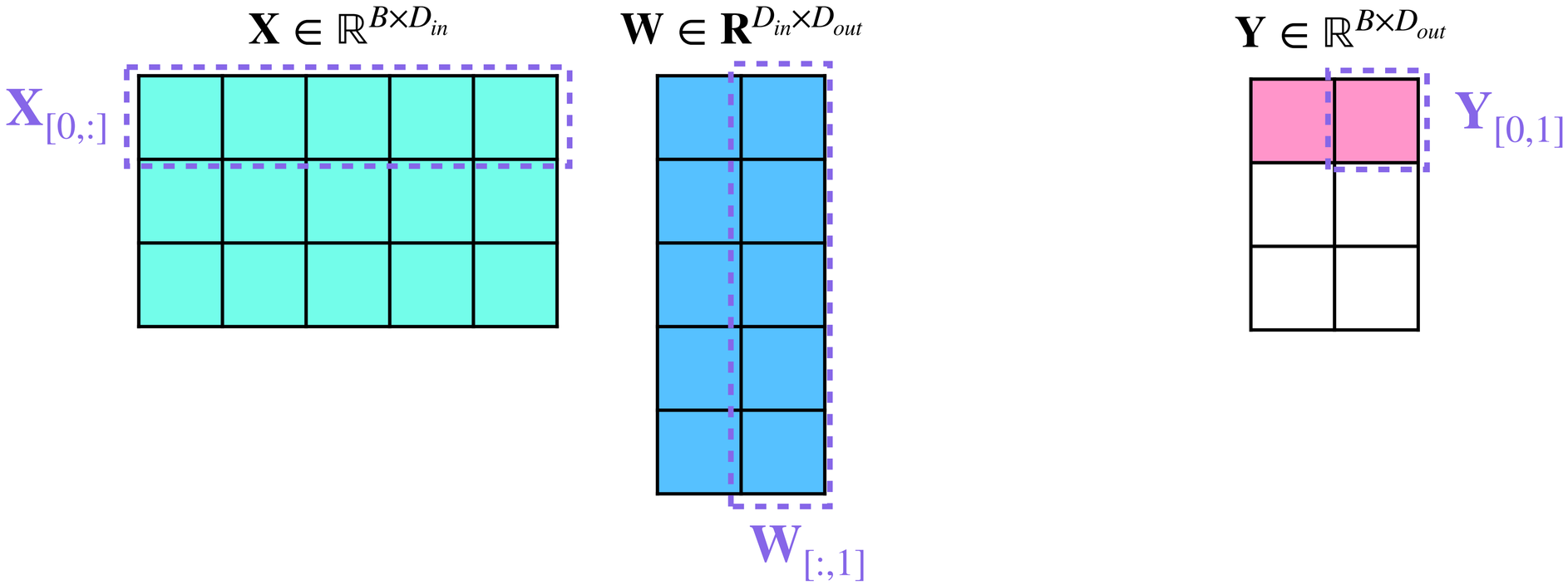
$\mathbf{Y} \in \mathbb{R}^{B \times D_{out}}$



A recap of matrix multiplication

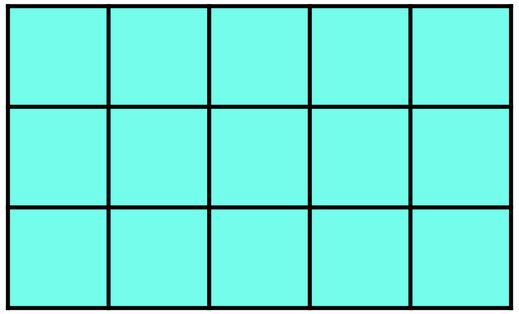


A recap of matrix multiplication

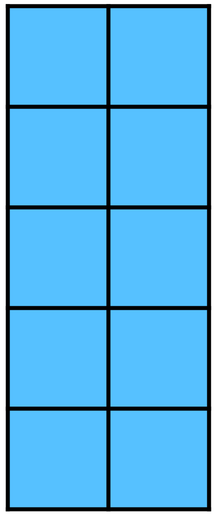


A recap of matrix multiplication

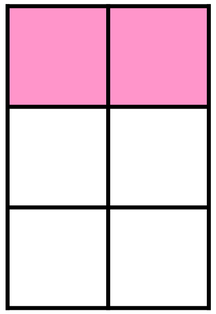
$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$



$\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$



$\mathbf{Y} \in \mathbb{R}^{B \times D_{out}}$



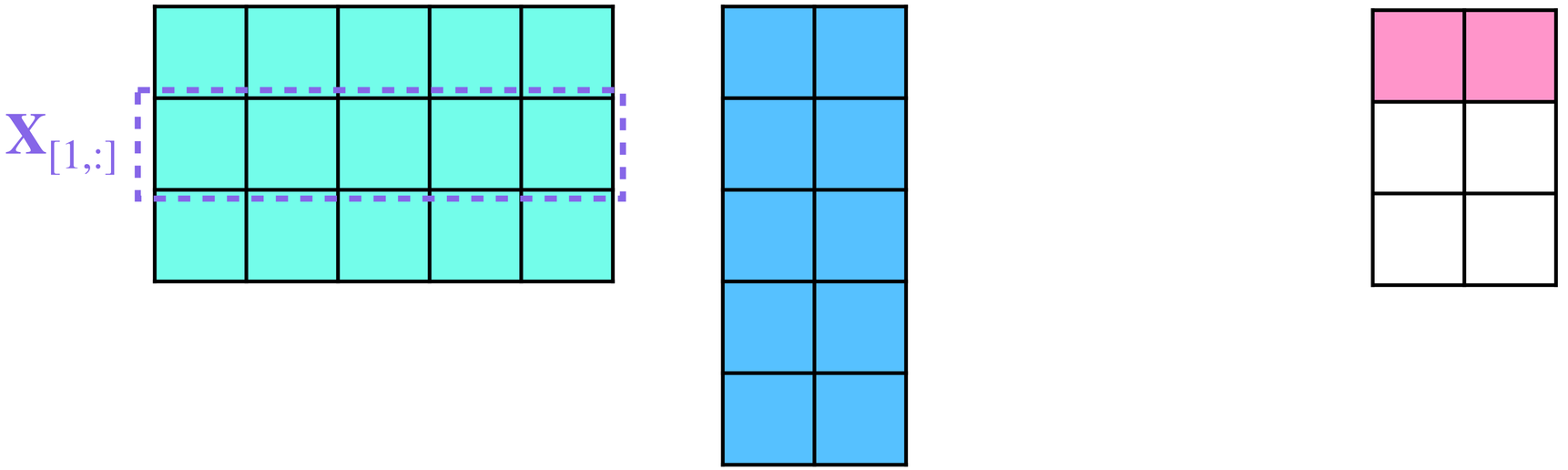
Repeat for the next prompt in the batch

A recap of matrix multiplication

$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$

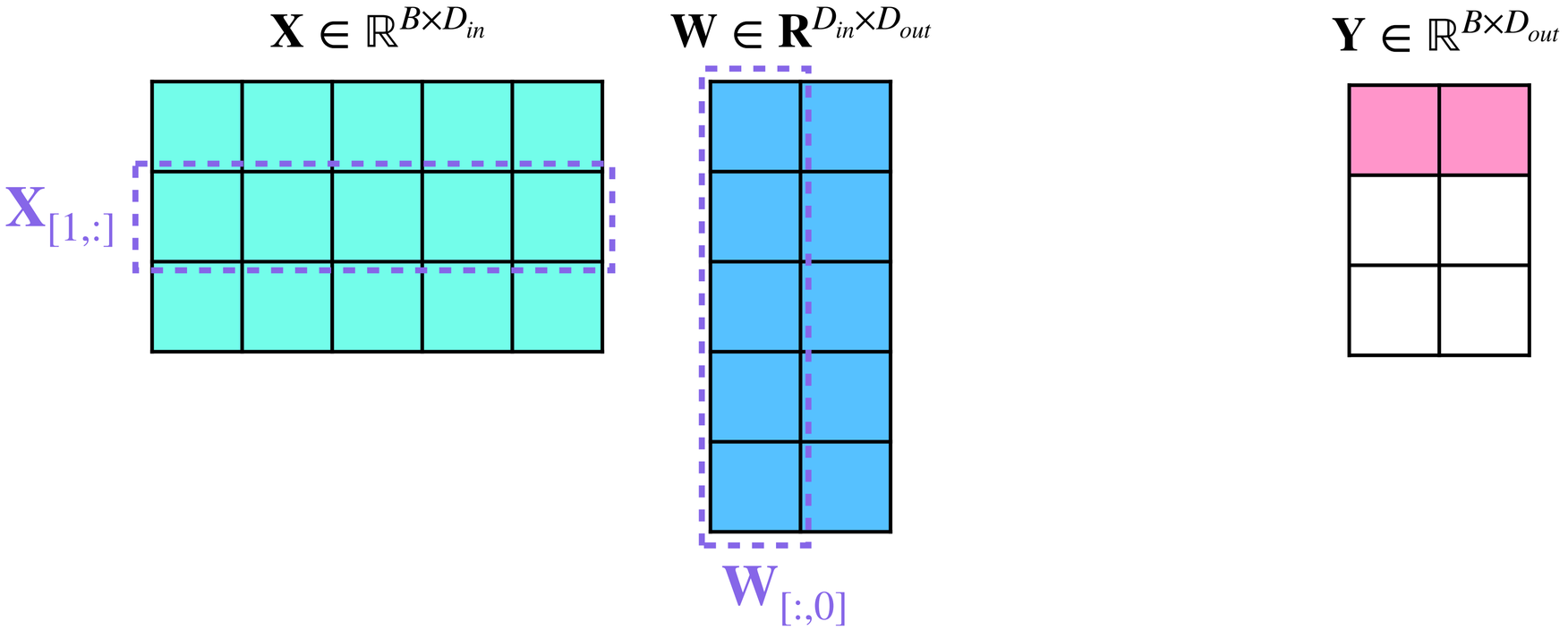
$\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$

$\mathbf{Y} \in \mathbb{R}^{B \times D_{out}}$



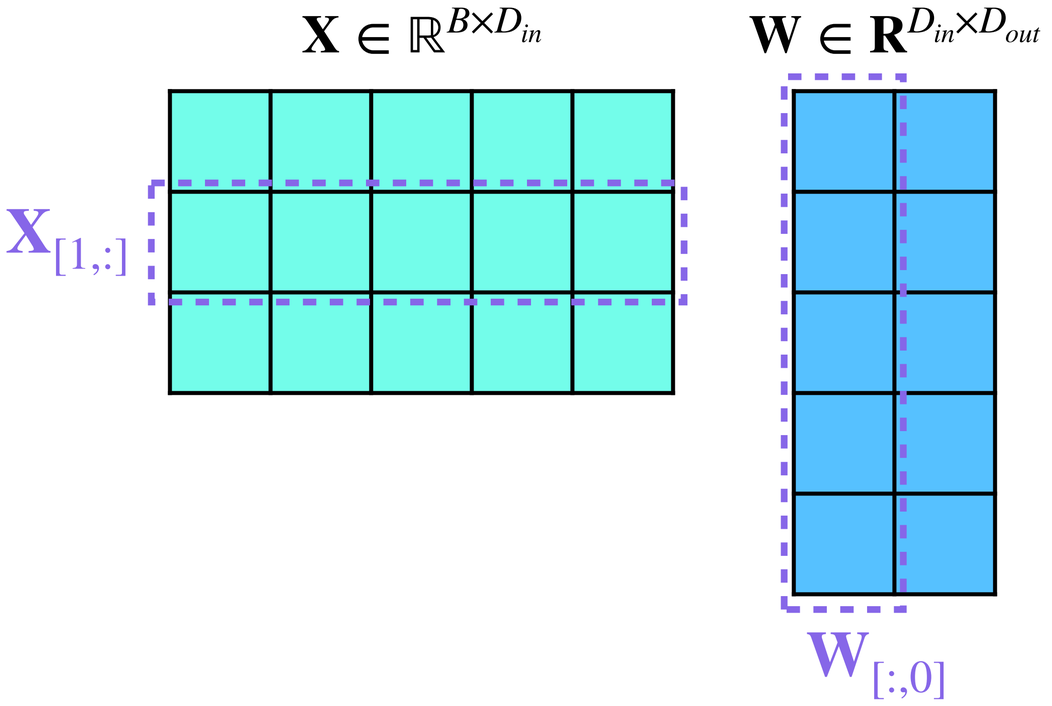
Repeat for the next prompt in the batch

A recap of matrix multiplication



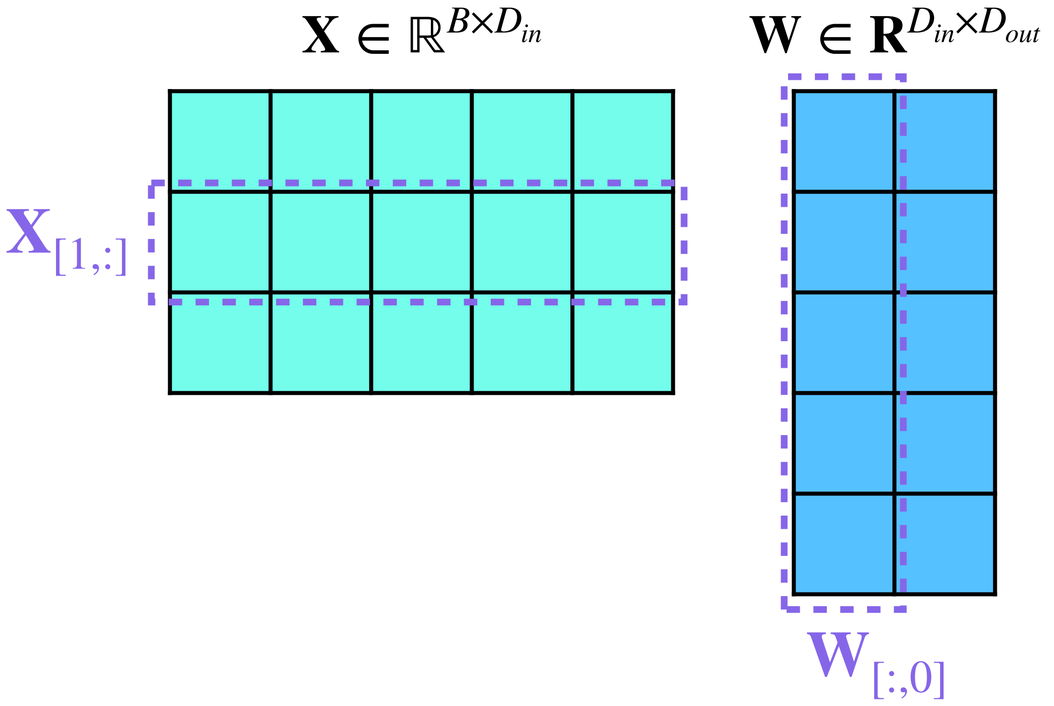
Repeat for the next prompt in the batch

A recap of matrix multiplication



Repeat for the next prompt in the batch

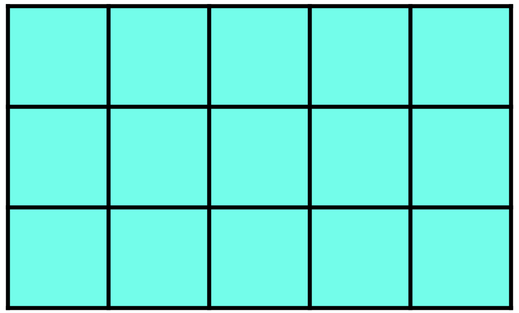
A recap of matrix multiplication



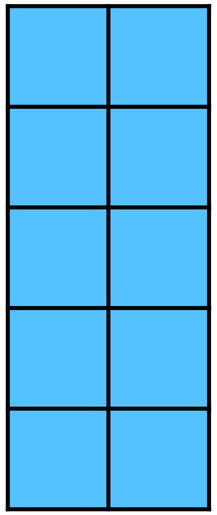
Repeat for the next prompt in the batch

A recap of matrix multiplication

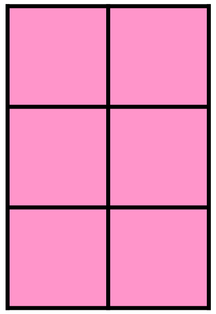
$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$



$\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$



$\mathbf{Y} \in \mathbb{R}^{B \times D_{out}}$

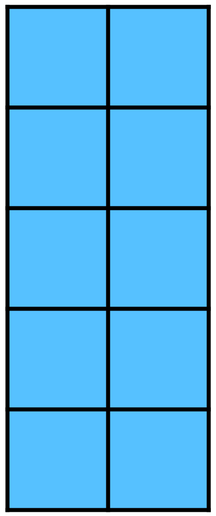


Back to quantization

Quantize weights only

int8

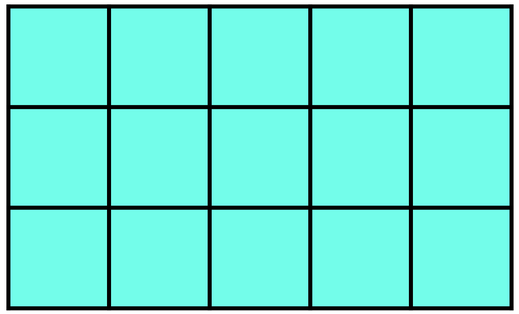
$$\hat{\mathbf{W}} \in \mathbf{R}^{D_{in} \times D_{out}}$$



Quantize weights only

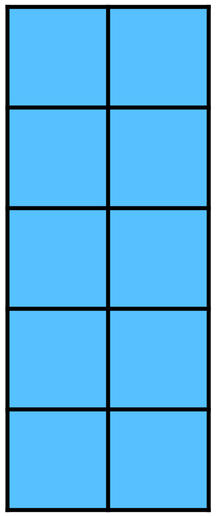
bf16

$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$



int8

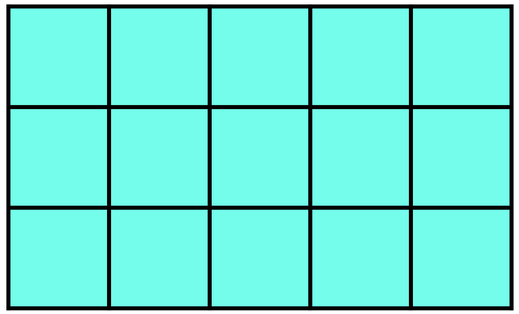
$\hat{\mathbf{W}} \in \mathbf{R}^{D_{in} \times D_{out}}$



Quantize weights only

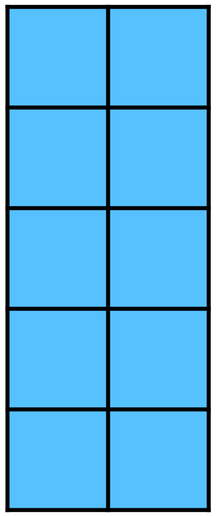
bf16

$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$



int8

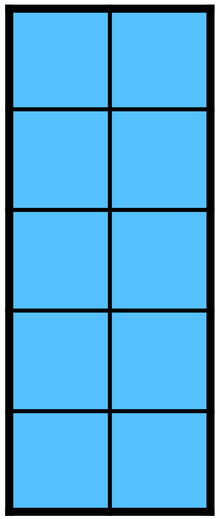
$\hat{\mathbf{W}} \in \mathbf{R}^{D_{in} \times D_{out}}$



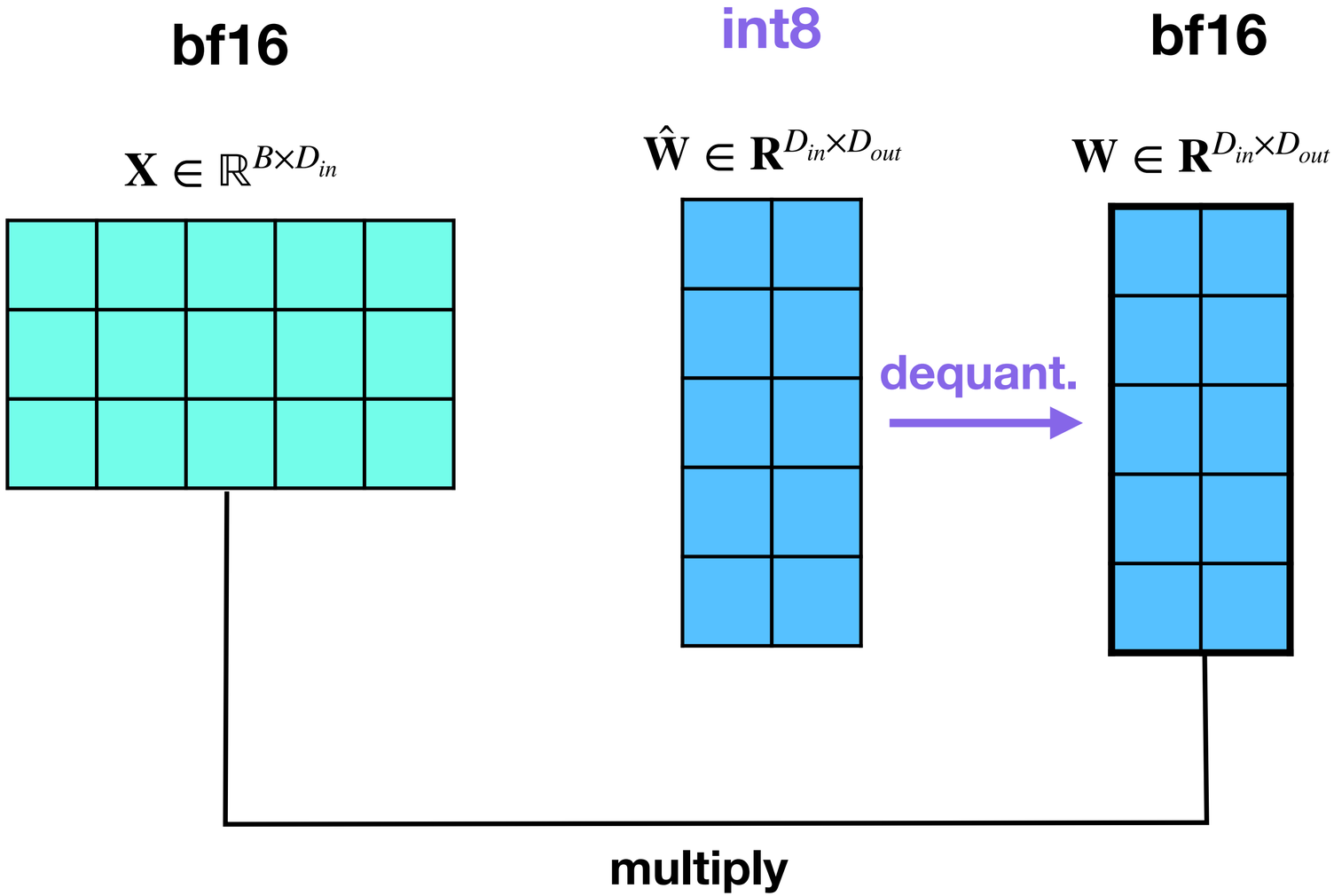
dequant.

bf16

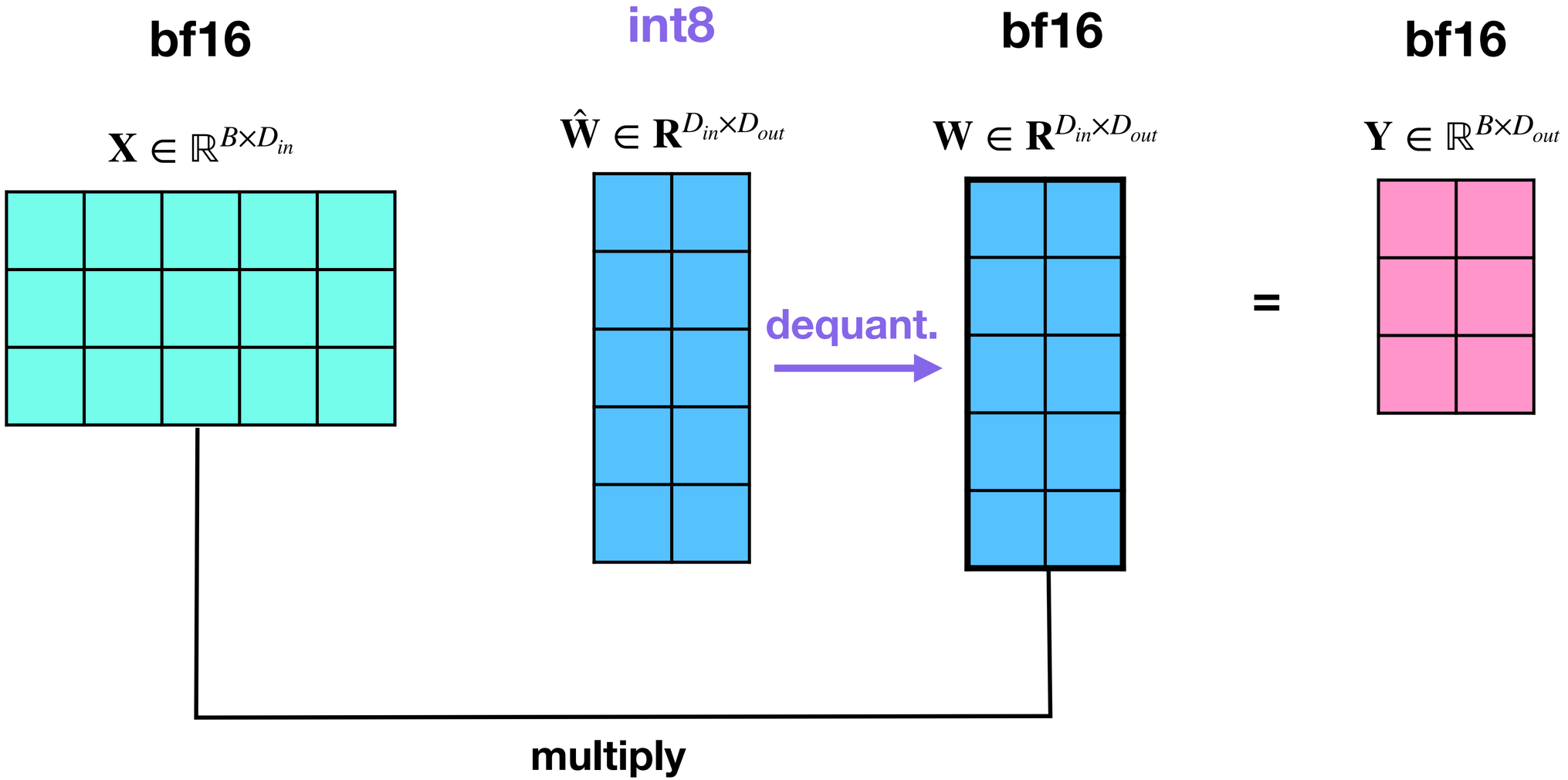
$\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$



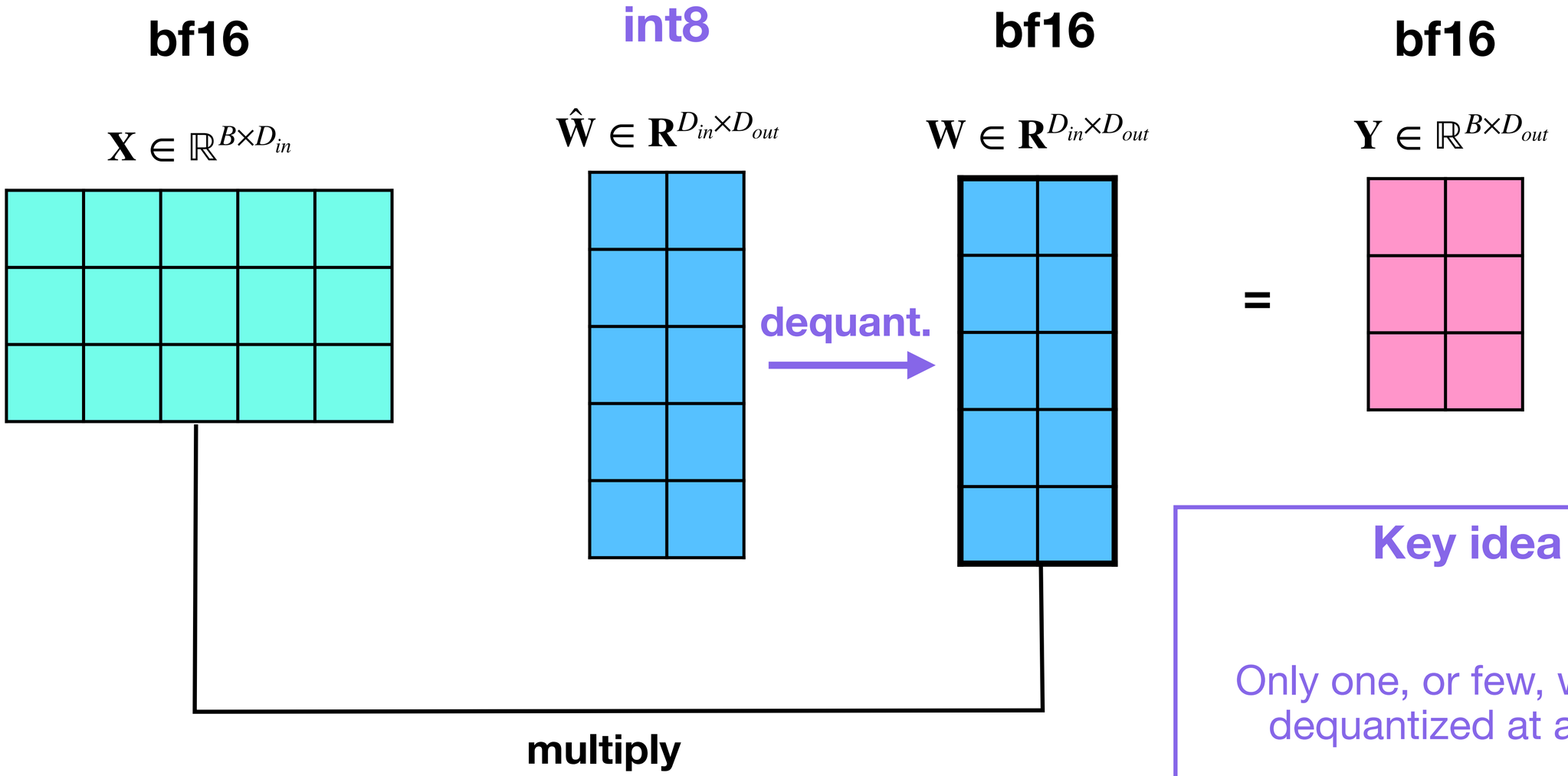
Quantize weights only



Quantize weights only



Quantize weights only



Key idea

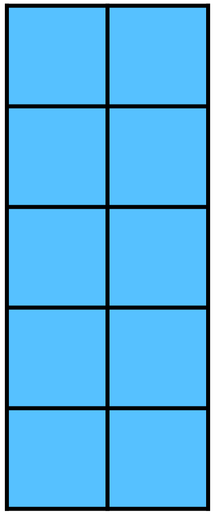
Only one, or few, weights dequantized at a time

Everything else remains in int8

Also quantize activations

int8

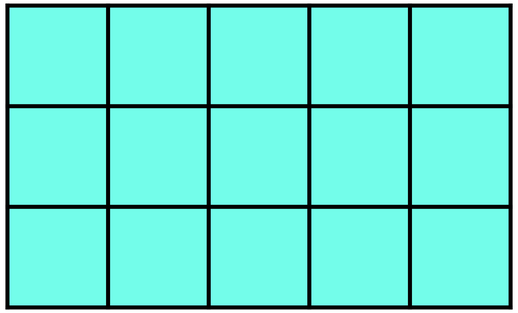
$$\hat{\mathbf{W}} \in \mathbf{R}^{D_{in} \times D_{out}}$$



Also quantize activations

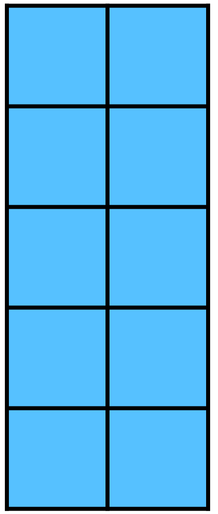
bf16

$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$



int8

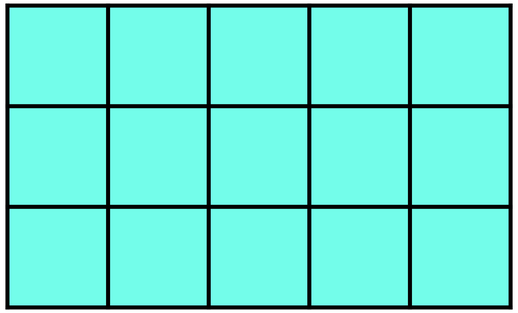
$\hat{\mathbf{W}} \in \mathbf{R}^{D_{in} \times D_{out}}$



Also quantize activations

bf16

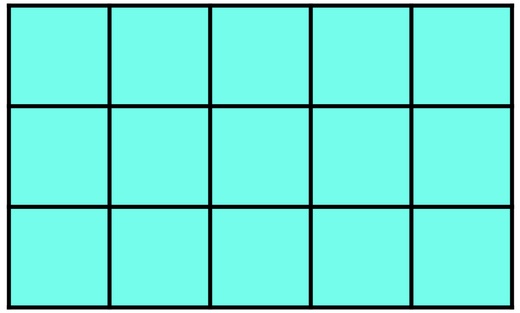
$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$



quantize

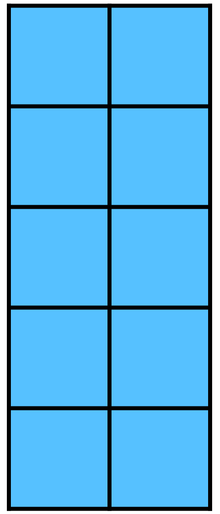
int8

$\hat{\mathbf{X}} \in \mathbb{R}^{B \times D_{in}}$



int8

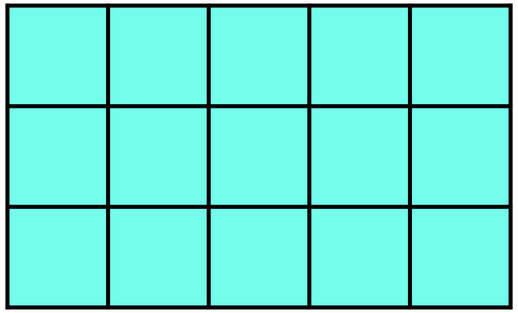
$\hat{\mathbf{W}} \in \mathbf{R}^{D_{in} \times D_{out}}$



Also quantize activations

bf16

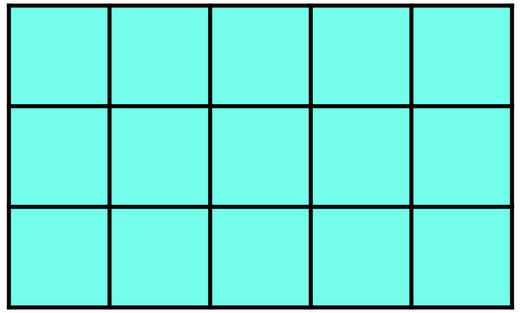
$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$



quantize

int8

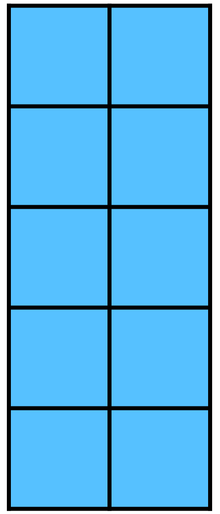
$\hat{\mathbf{X}} \in \mathbb{R}^{B \times D_{in}}$



x

int8

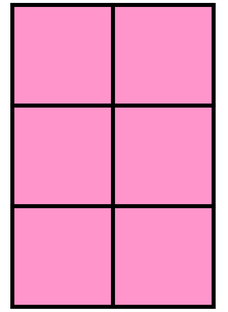
$\hat{\mathbf{W}} \in \mathbf{R}^{D_{in} \times D_{out}}$



=

int8

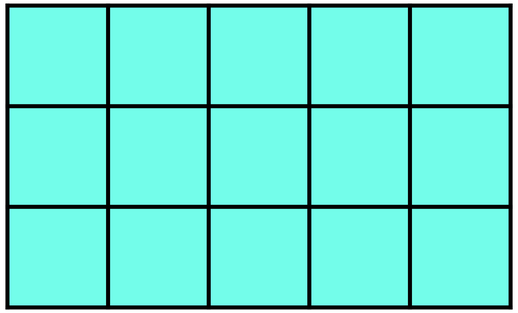
$\hat{\mathbf{Y}} \in \mathbb{R}^{B \times D_{out}}$



Also quantize activations

bf16

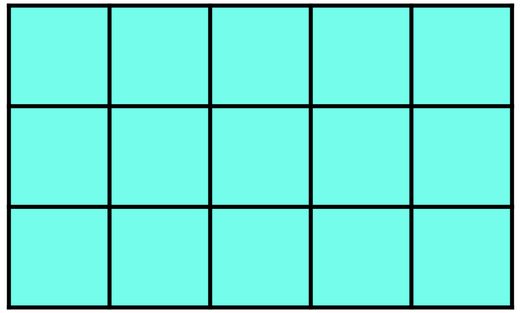
$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$



quantize

int8

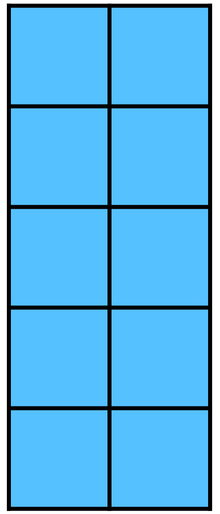
$\hat{\mathbf{X}} \in \mathbb{R}^{B \times D_{in}}$



x

int8

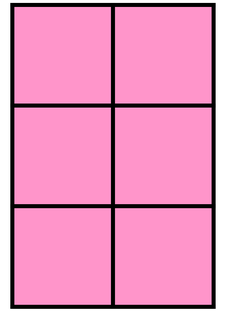
$\hat{\mathbf{W}} \in \mathbf{R}^{D_{in} \times D_{out}}$



=

int8

$\hat{\mathbf{Y}} \in \mathbb{R}^{B \times D_{out}}$



Key idea

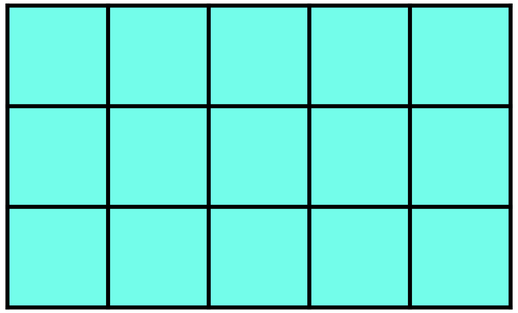
All operations in int8

GPU can leverage custom integer kernels

Also quantize activations

bf16

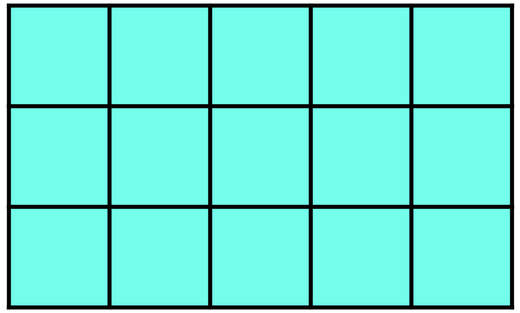
$\mathbf{X} \in \mathbb{R}^{B \times D_{in}}$



quantize

int8

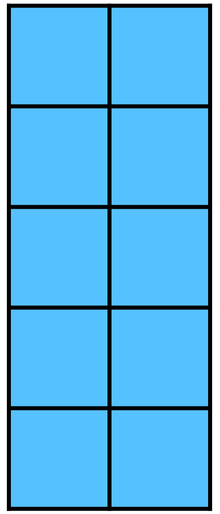
$\hat{\mathbf{X}} \in \mathbb{R}^{B \times D_{in}}$



x

int8

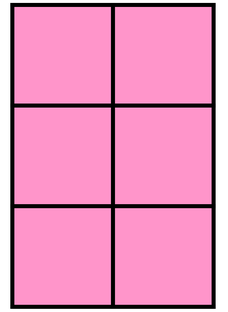
$\hat{\mathbf{W}} \in \mathbf{R}^{D_{in} \times D_{out}}$



=

int8

$\hat{\mathbf{Y}} \in \mathbb{R}^{B \times D_{out}}$



Note

- ˆ in $\hat{\mathbf{X}}$, $\hat{\mathbf{W}}$ and $\hat{\mathbf{Y}}$ denotes the quantized version
- $\mathbf{X}$ and $\mathbf{Y}$ are both activations
- Input of one layer L, that is, $\mathbf{X}$, is output of layer L-1

Key idea

All operations in int8

GPU can leverage custom integer kernels

Weight + Activation quantization

- Activation quantization adds more complications

Weight + Activation quantization

- Activation quantization adds more complications
- Weights are fixed, that is, they do not change once we are done training
  - Quantization scale s needs to be learnt only once
- Activations change with every new prompt
  - Need to set the quantize scale s for every new input

$$\gamma = \frac{2^{N-1}}{\max(|\mathbf{W}|)}$$

Fixed per model

$$\gamma = \frac{2^{N-1}}{\max(|\mathbf{X}|)}$$

Change for every prompt

Weight + Activation quantization

- Activation quantization adds more complications
- Weights are fixed, that is, they do not change once we are done training
  - Quantization scale s needs to be learnt only once
- Activations change with every new prompt
  - Need to set the quantize scale s for every new input
- Dynamic quantization
  - Learn the scale for every input on the fly
  - Accurate but slow (have to re-compute scale for every input)

$$\gamma = \frac{2^{N-1}}{\max(|\mathbf{W}|)}$$

Fixed per model

$$\gamma = \frac{2^{N-1}}{\max(|\mathbf{X}|)}$$

Change for every prompt

Weight + Activation quantization

- Activation quantization adds more complications
- Weights are fixed, that is, they do not change once we are done training
  - Quantization scale s needs to be learnt only once

$$\gamma = \frac{2^{N-1}}{\max(|\mathbf{W}|)}$$

Fixed per model

- Activations change with every new prompt
  - Need to set the quantize scale s for every new input

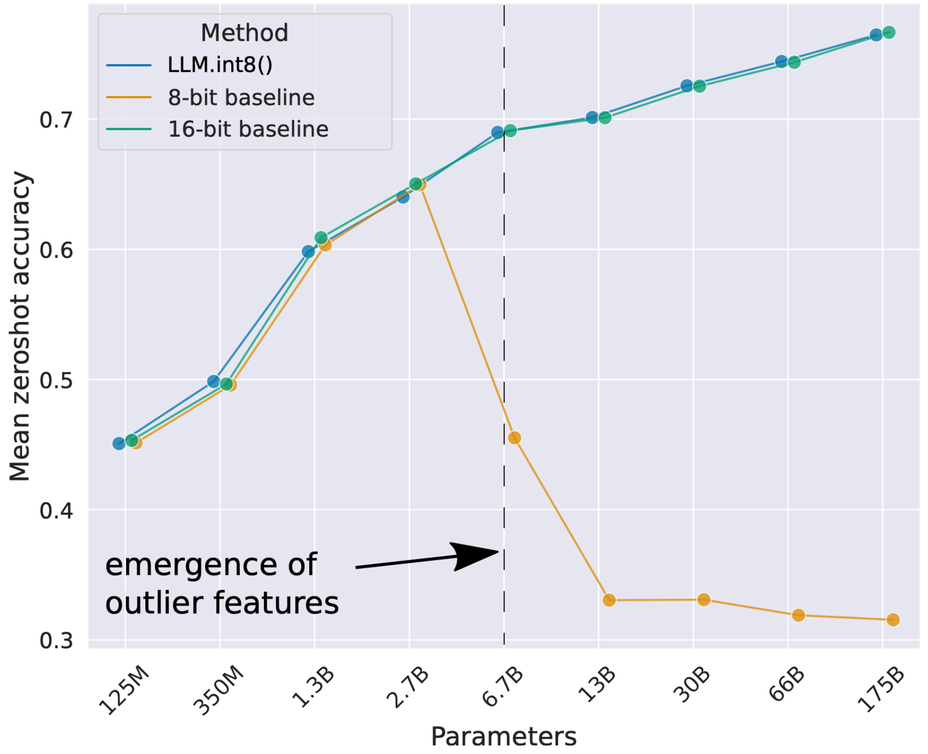
$$\gamma = \frac{2^{N-1}}{\max(|\mathbf{X}|)}$$

Change for every prompt

- Dynamic quantization
  - Learn the scale for every input on the fly
  - Accurate but slow (have to re-compute scale for every input)
- Static quantization
  - Use a calibration dataset (e.g., few hundred samples) to learn the scale
  - Fast but less accurate

LLM.int8() quantization

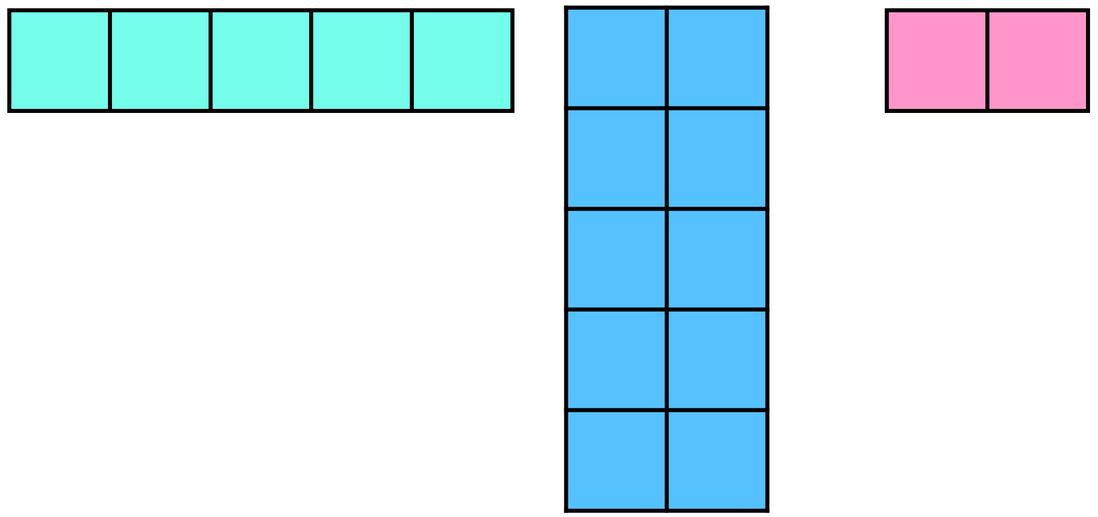
- Available in HuggingFace via the bitsandbytes library
- Outliers emerge after model size exceeds ~3B
- Luckily, outliers concentrated in few channels
- **Key idea:** Mixed precision quantization
  - Do not quantize outliers (less than 1% of the weights)
  - All other weights get quantized to int8



[Dettmers et al]

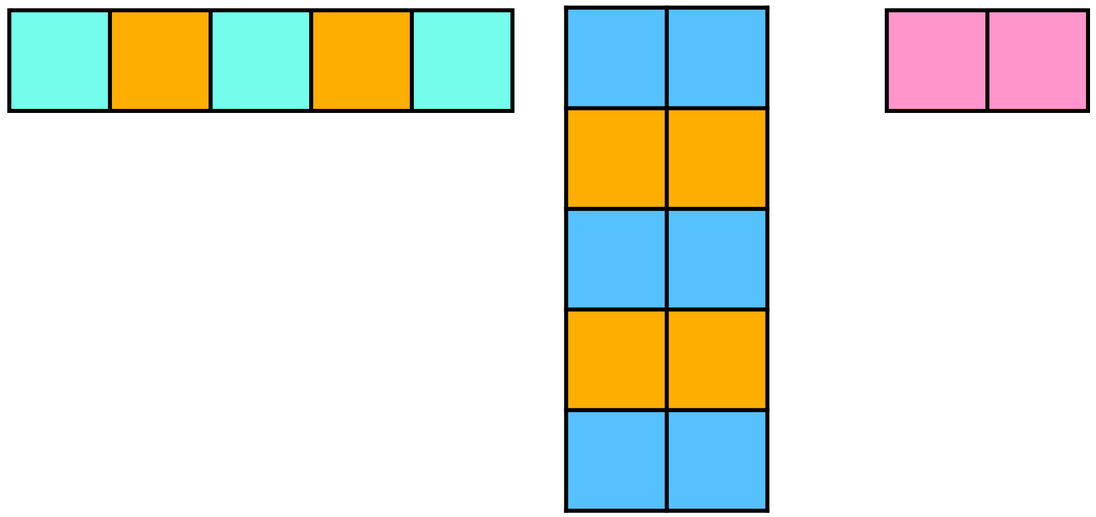
Mixed precision quantization

$\mathbf{X} \in \mathbb{R}^{D_{in}}$ $\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$ $\mathbf{Y} \in \mathbb{R}^{D_{out}}$



Mixed precision quantization

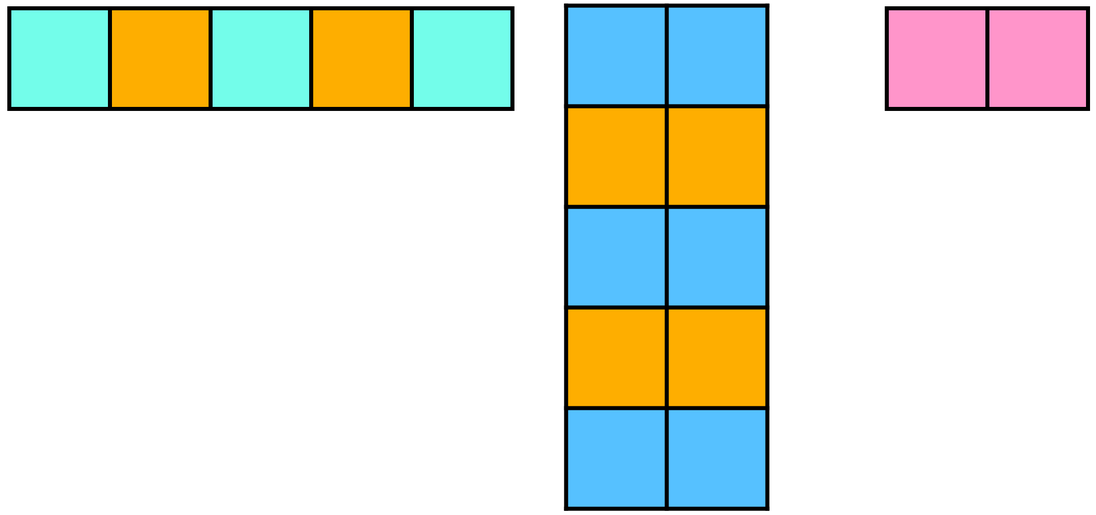
$\mathbf{X} \in \mathbb{R}^{D_{in}}$ $\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$ $\mathbf{Y} \in \mathbb{R}^{D_{out}}$



Outlier activations and corresponding weights. E.g., all activations are max. 1.1 but outliers are >10

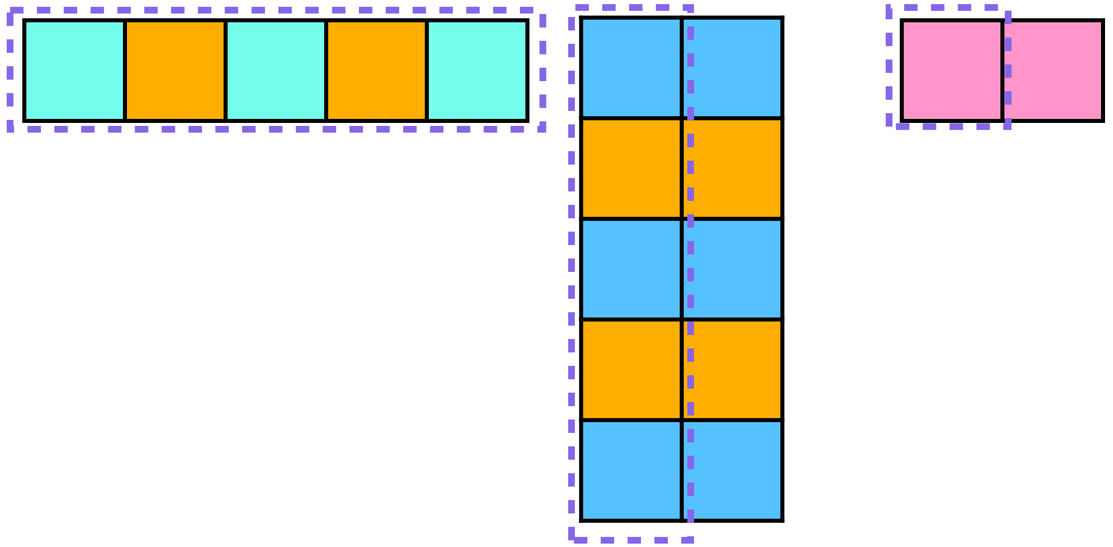
Mixed precision quantization

$\mathbf{X} \in \mathbb{R}^{D_{in}}$ $\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$ $\mathbf{Y} \in \mathbb{R}^{D_{out}}$

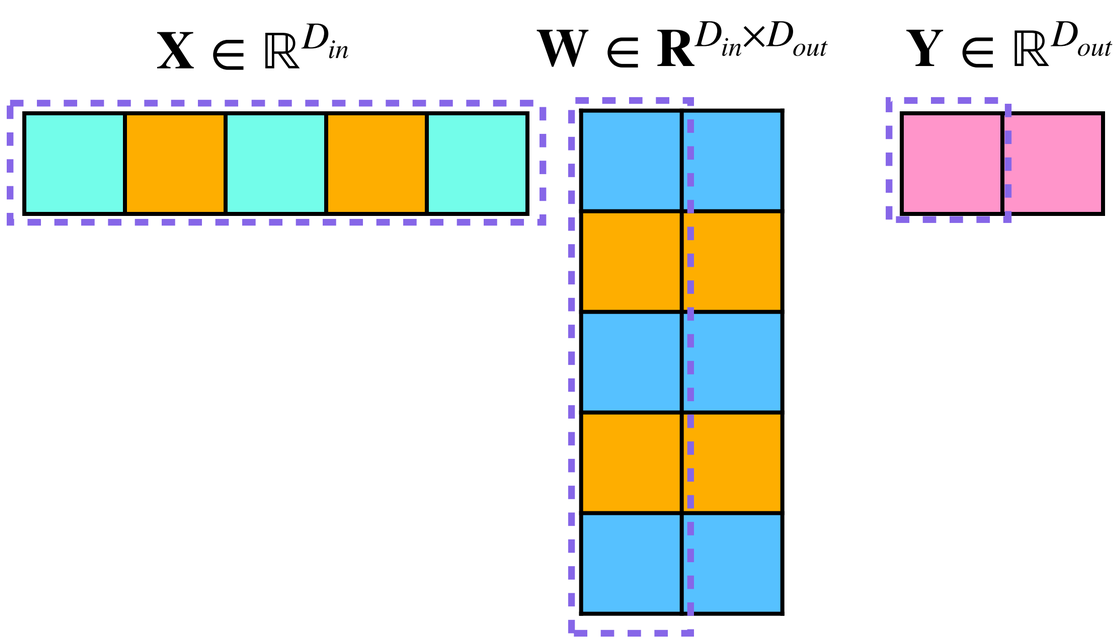


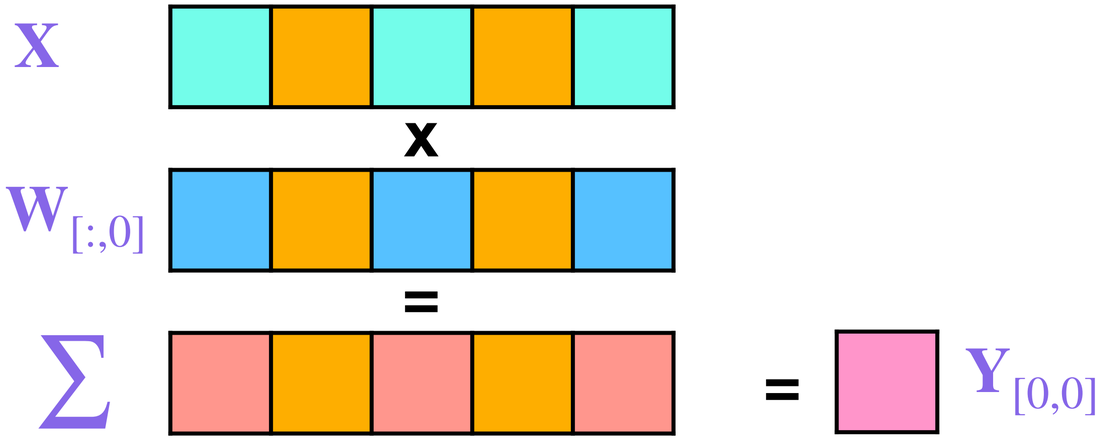
Mixed precision quantization

$\mathbf{X} \in \mathbb{R}^{D_{in}}$ $\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$ $\mathbf{Y} \in \mathbb{R}^{D_{out}}$

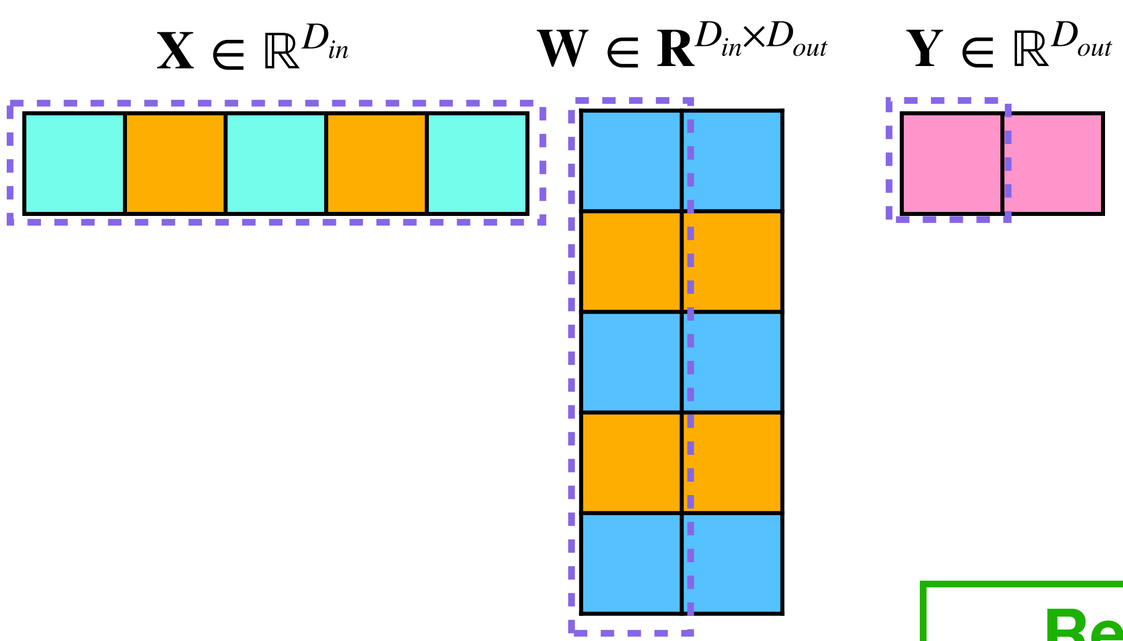


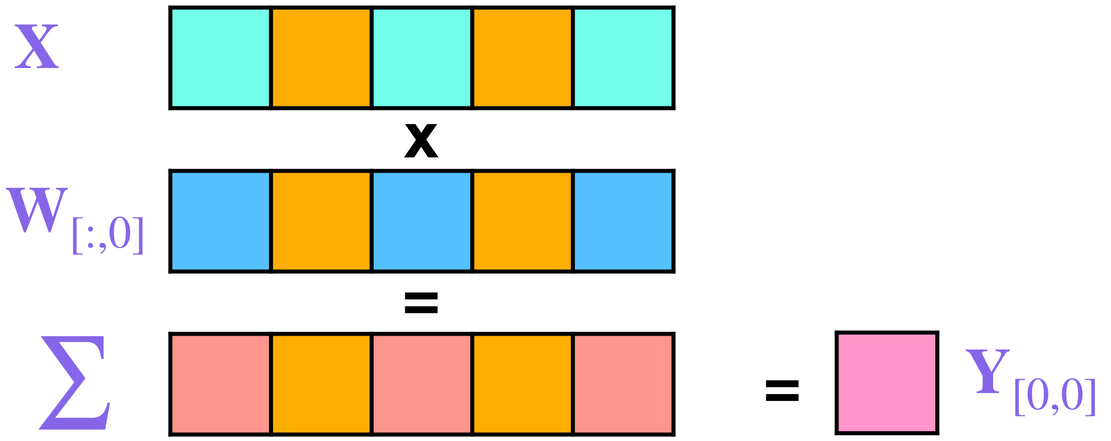
Mixed precision quantization



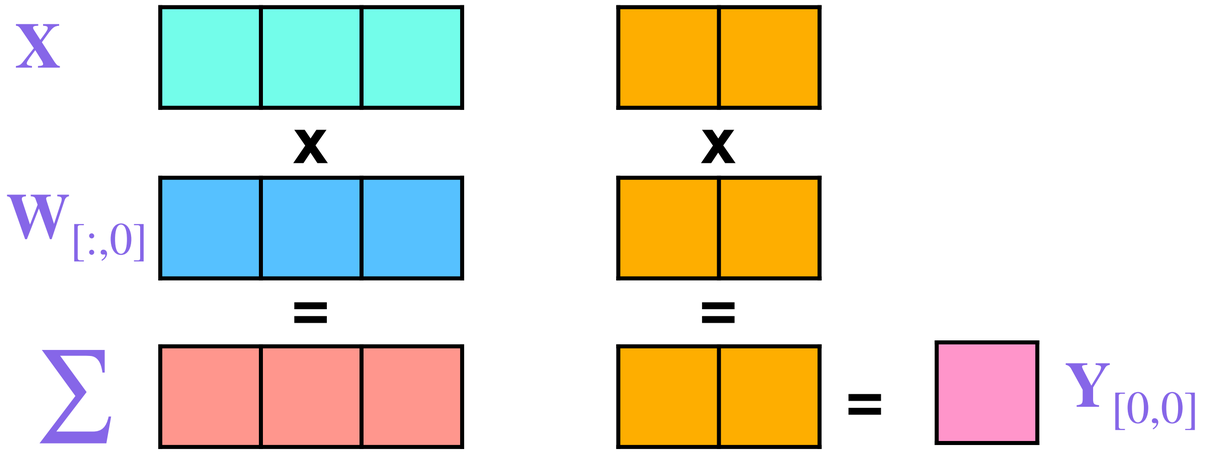


Mixed precision quantization

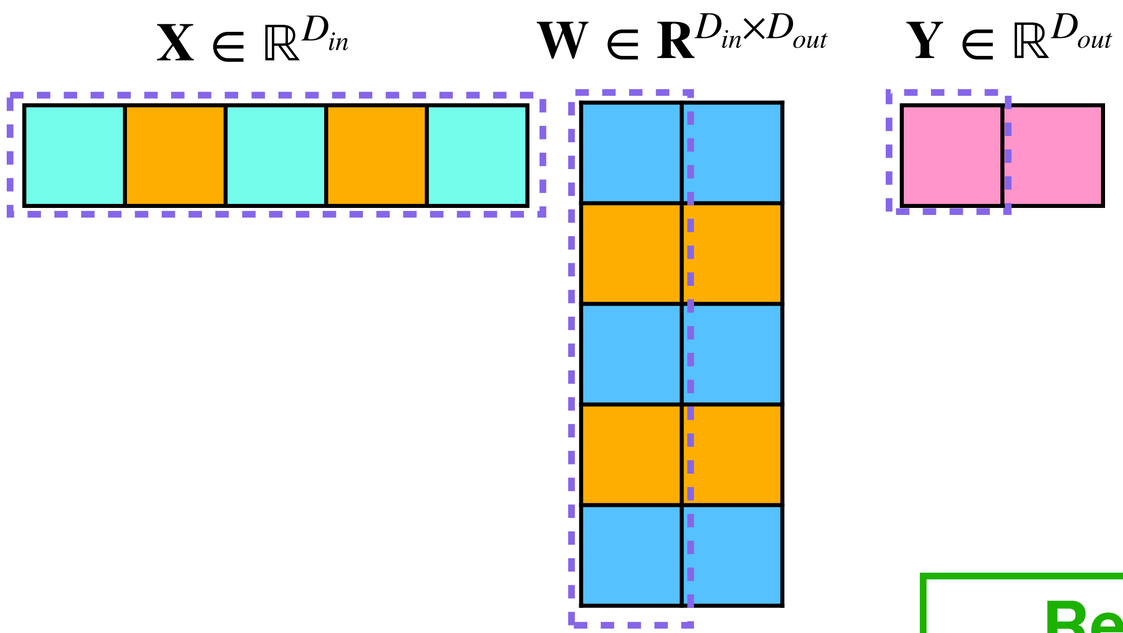


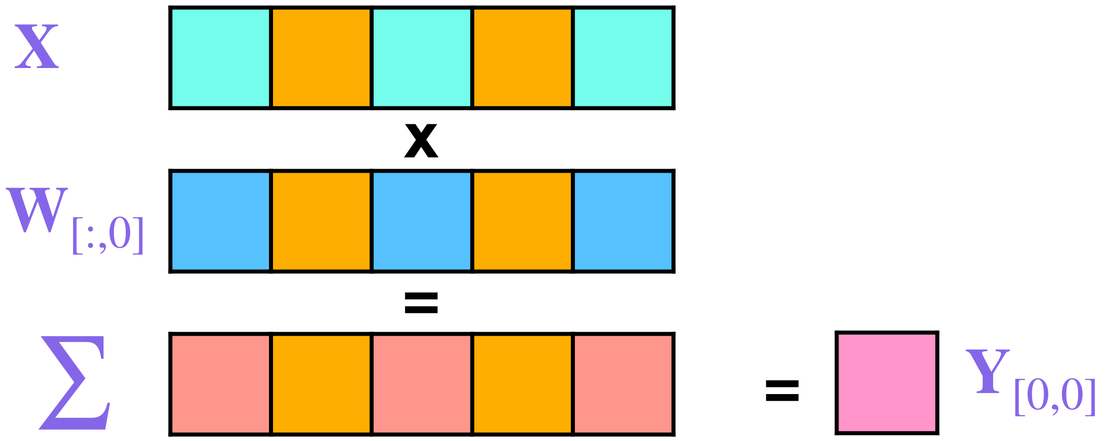


Re-arrange No effect on output

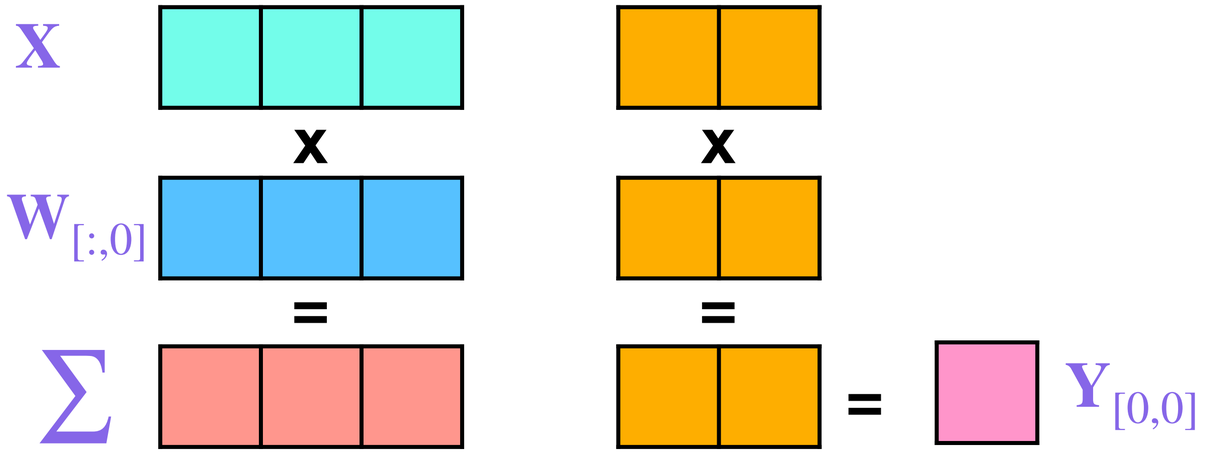


Mixed precision quantization





Re-arrange No effect on output



Quantize to int8

Keep in bf16

LLM.int8(): Pros and cons

- Simple to implement
- Works on the fly
- Need to set the outlier threshold manually (e.g., 6)
- Paper shows that the runtime is slower than the base model
- May not work on all hardware — bitsandbytes works on GPUs only (so far)

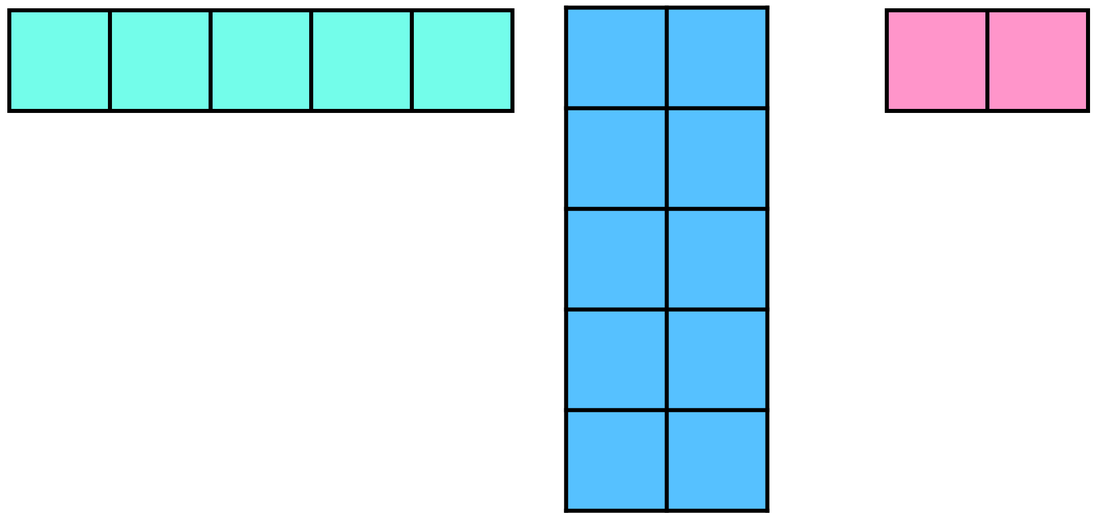
GPTQ

- LLM.int8() performs poorly when scaled to 4 bits
- GPTQ aims to quantize to lower than 8-bit precision, e.g., 4 or 2 bits
- **Key idea:** Minimize the loss between the output of quantized and unquantizd matmul

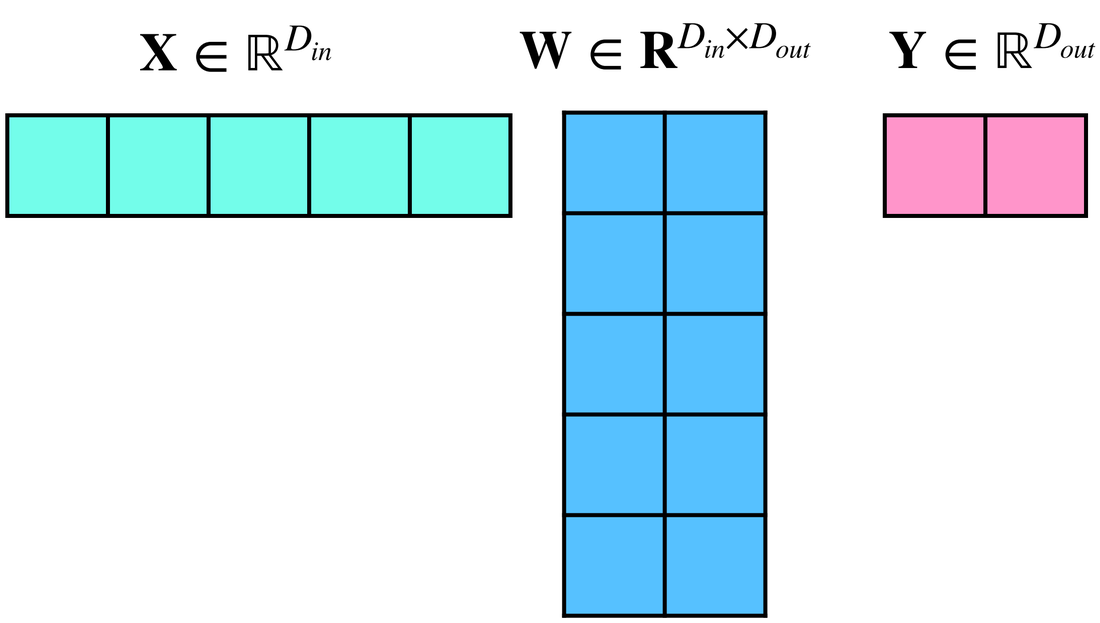
$$\hat{\mathbf{W}} = \arg\min ||\mathbf{XW} - \mathbf{X}\hat{\mathbf{W}}||_2^2$$

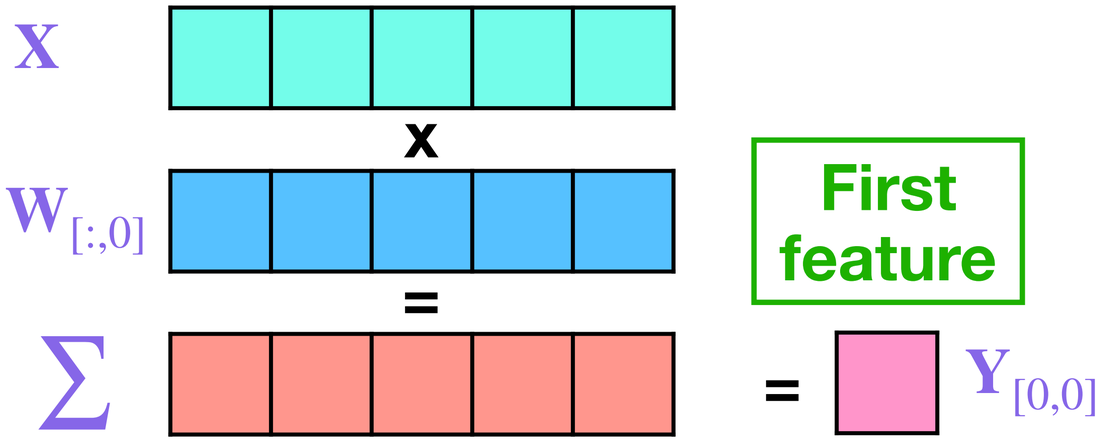
GPTQ: Performing quantization in parallel

$\mathbf{X} \in \mathbb{R}^{D_{in}}$ $\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$ $\mathbf{Y} \in \mathbb{R}^{D_{out}}$

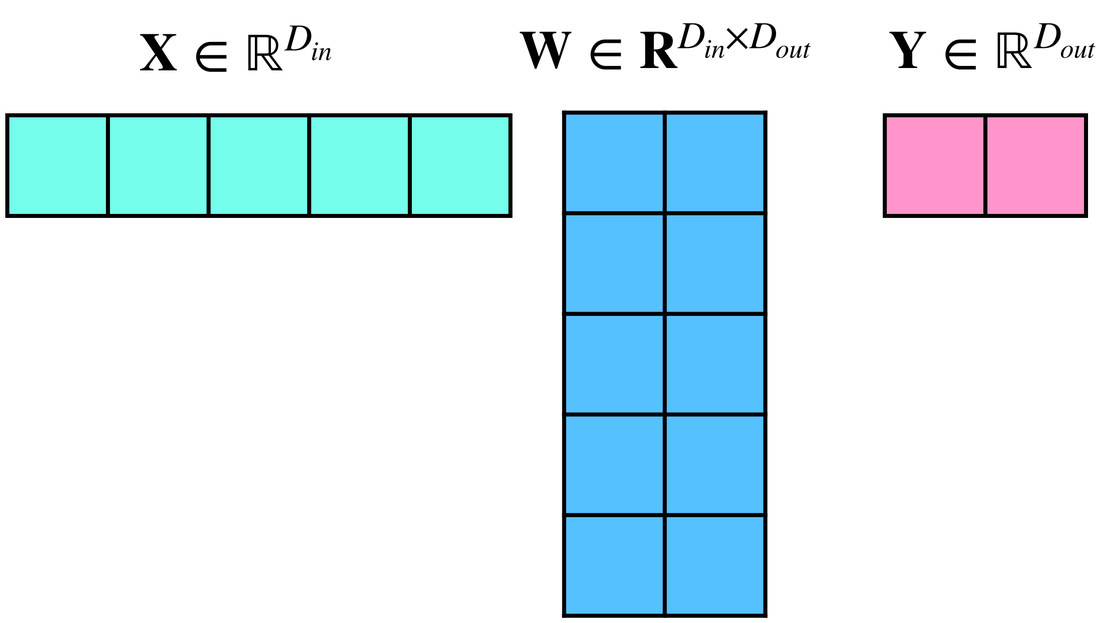


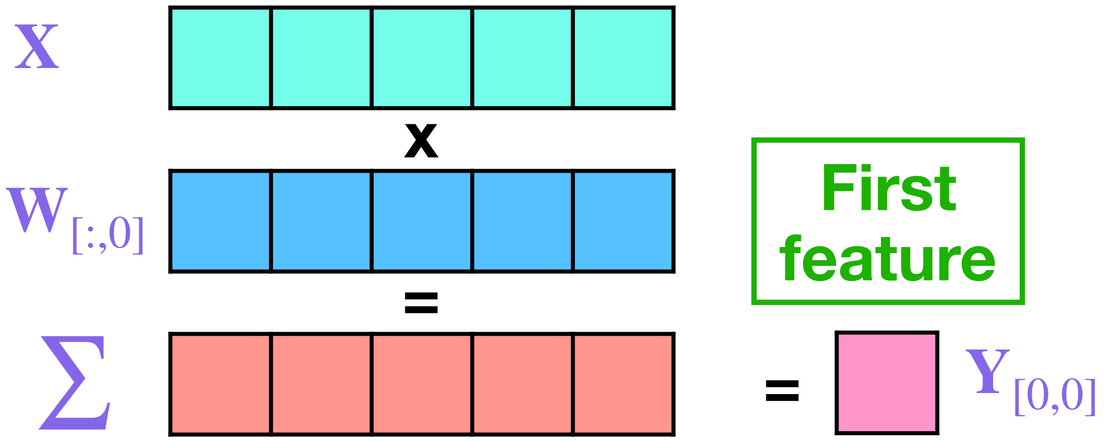
GPTQ: Performing quantization in parallel

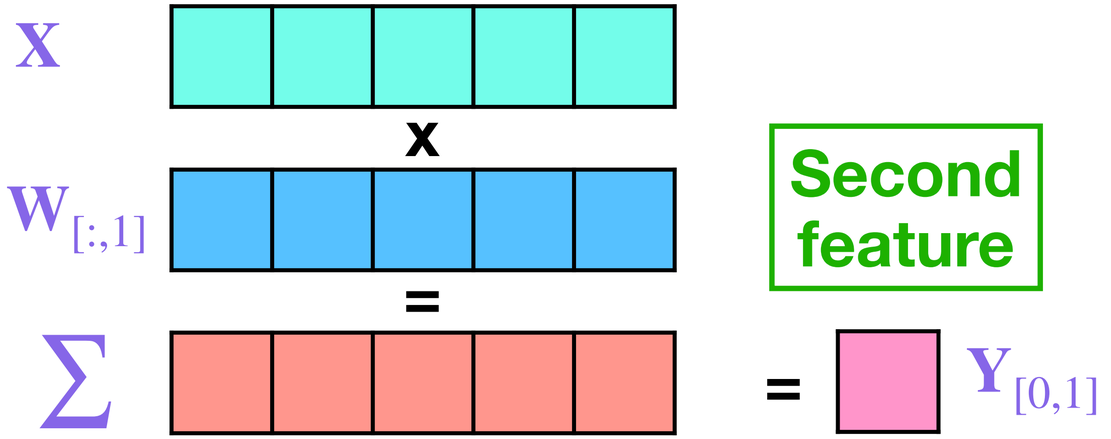




GPTQ: Performing quantization in parallel

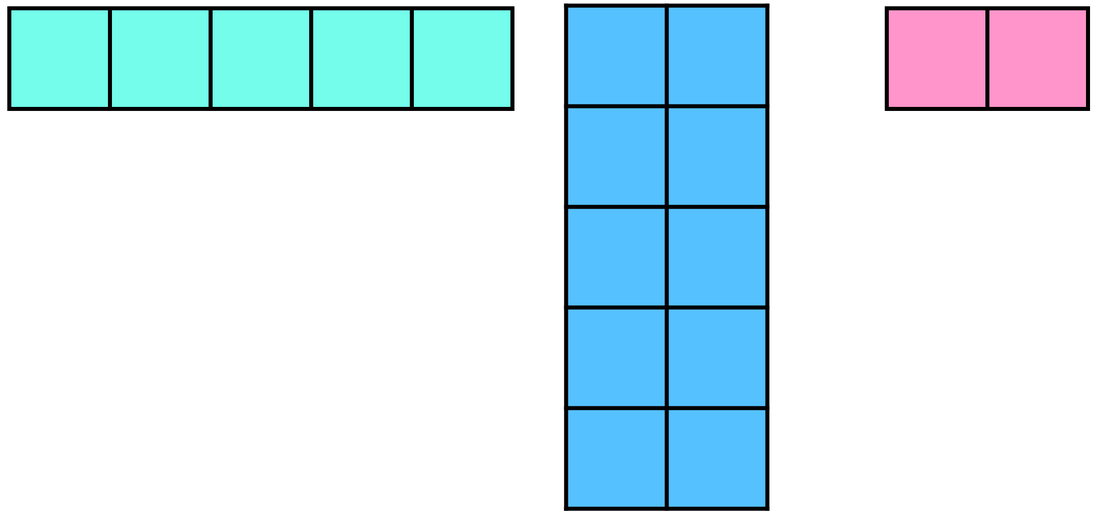


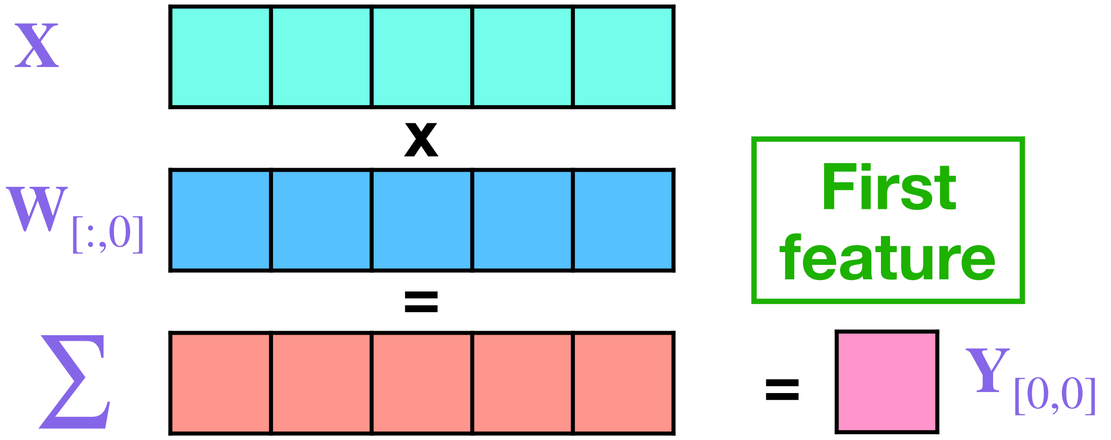




GPTQ: Performing quantization in parallel

$\mathbf{X} \in \mathbb{R}^{D_{in}}$ $\mathbf{W} \in \mathbf{R}^{D_{in} \times D_{out}}$ $\mathbf{Y} \in \mathbb{R}^{D_{out}}$

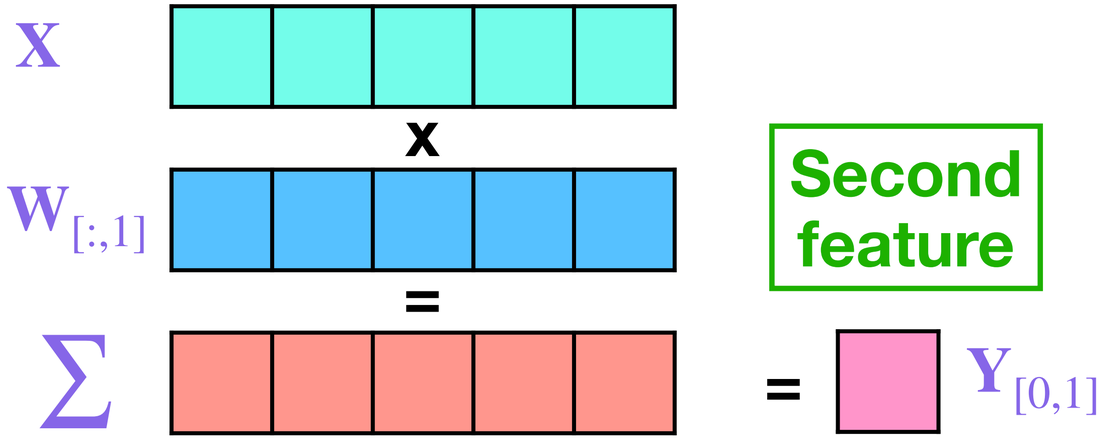




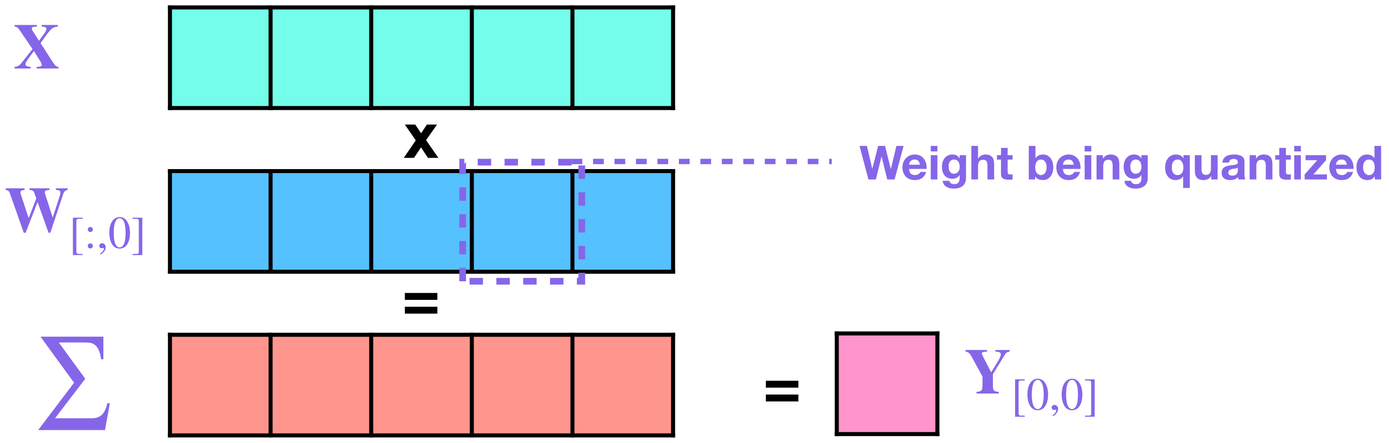
Can perform the computation for different columns **in parallel**

$$\hat{\mathbf{W}} = \arg\min ||\mathbf{XW} - \mathbf{X}\hat{\mathbf{W}}||_2^2$$

$$= \arg\min \sum ||\mathbf{XW}_{[:,i]} - \mathbf{X}\hat{\mathbf{W}}_{[:,i]}||_2^2$$



Iteratively quantizing weights



- Quantize this weight
- Update the unquantized weights in $\mathbf{W}_{[:,0]}$ to minimize $||\mathbf{XW} - \mathbf{X}\hat{\mathbf{W}}||_2^2$
- Repeat for all weights in $\mathbf{W}_{[:,0]}$
- Recursively repeat for all columns

GPTQ: Pros and cons

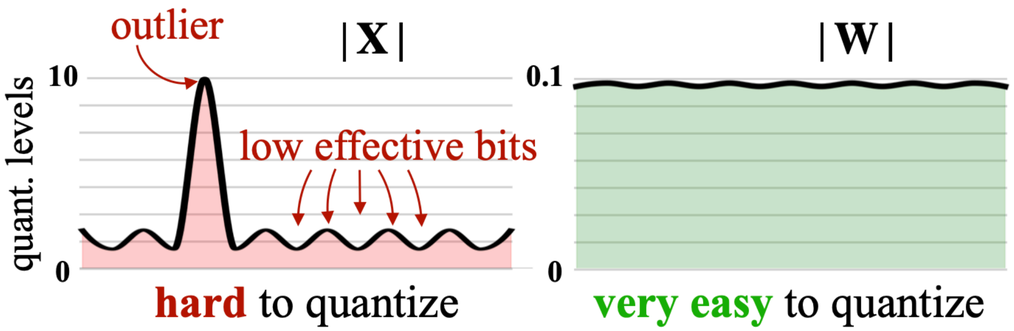
- Quantizes models very quickly by using second order optimization
- Only quantizes weights so runs on the fly
- Needs a calibration dataset
- Unlike LLM.int8(), does not need to guess the outlier values, instead, works based off the data

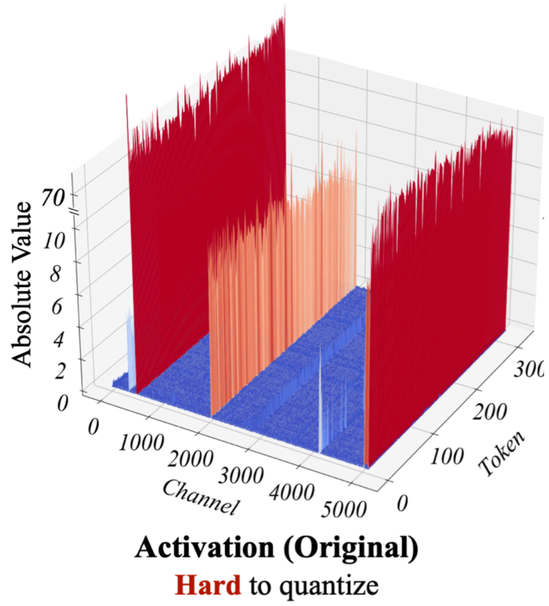
Exercise

SmoothQuant

- **Intuition:** No outliers in weights, many large outliers in activations
  - Weights: No outliers means we use the quantization range efficiently (many effective bits)
  - Activations: Large outliers means we use the quantization range inefficiently (few effective bits)

- However, outliers are concentrated in a few channels

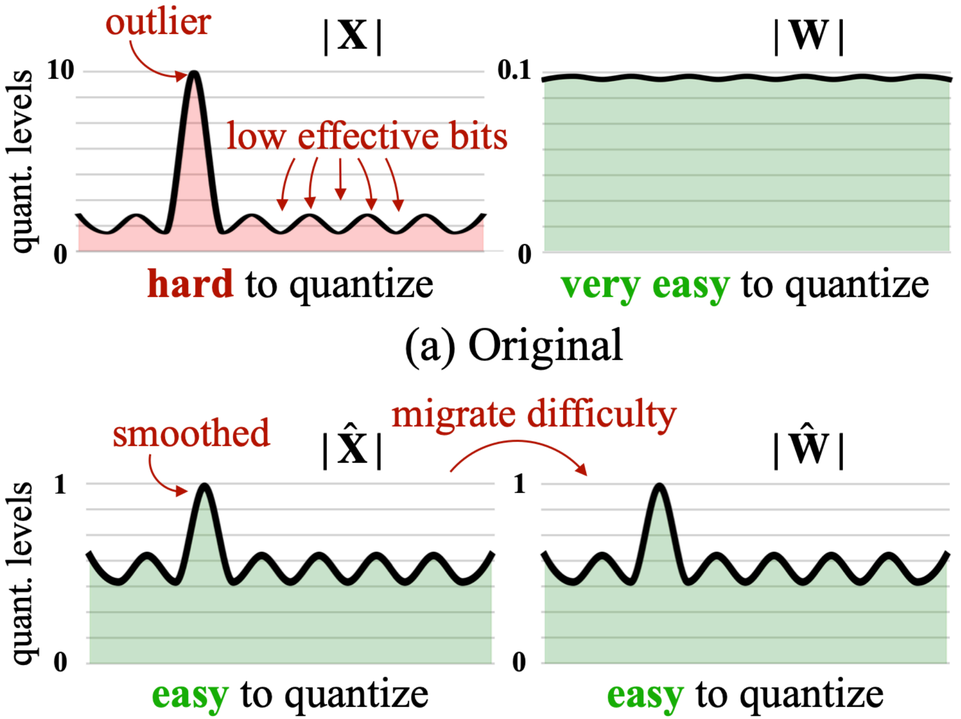




[Xiao et al.]

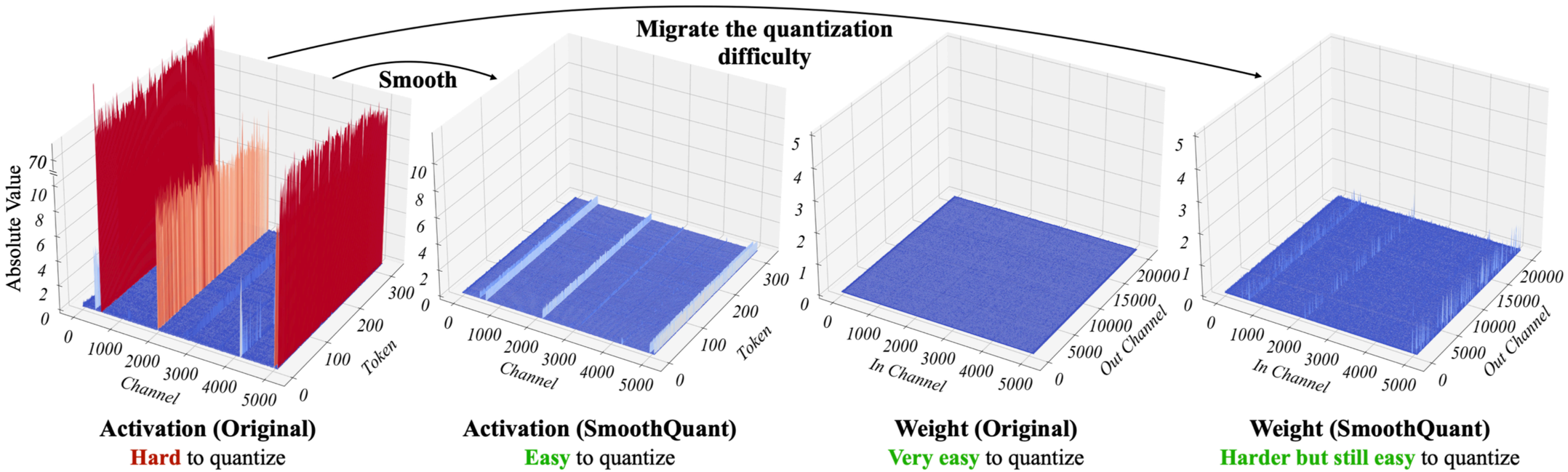
SmoothQuant

- **Key idea:** Migrating outliers between activations and weights
  - Smooth out the activations by dividing them with a channel-wise scaling term $\mathbf{s}$
  - Divide the weights by the same term to migrate the outliers
- $\bar{\mathbf{X}} = \mathbf{S}^{-1}\mathbf{X}$ with $\mathbf{X}, \mathbf{S}, \bar{\mathbf{X}} \in \mathbb{R}^{D_{in}}$ and $\mathbf{S} > 1$
- Obtain $\bar{\mathbf{W}}$ by multiplying with $\mathbf{S}$
- $\mathbf{X}\mathbf{W} = \bar{\mathbf{X}}\bar{\mathbf{W}}$
- $\mathbf{S}_j = \max(|\mathbf{X}_j|)^{\alpha} / \max(|\mathbf{W}_j|)^{1-\alpha}$
  - Pick $\alpha \in [0,1]$ based on the model / quantization
- Proceed with quantization



[Xiao et al.]

SmoothQuant



[Xiao et al.]

SmoothQuant: Pros and cons

- The smoothing process done offline so faster than dynamic quantization
- Needs a calibration dataset
- Can mix and match with downstream quantization strategies

AWQ

- Somewhat of a mix of ideas discussed earlier

- **Key ideas**
  - Weights and activations interact, so quantization should be mindful of that
  - Scale weights and activations before quantization
  - Learn the quantization parameters by solving an optimization problem over the data
  - Avoid mixed precision like mix of int8 and bf16

$$\mathbf{s}^* = \arg\min_{\mathbf{s}} \mathcal{L}(\mathbf{s})$$

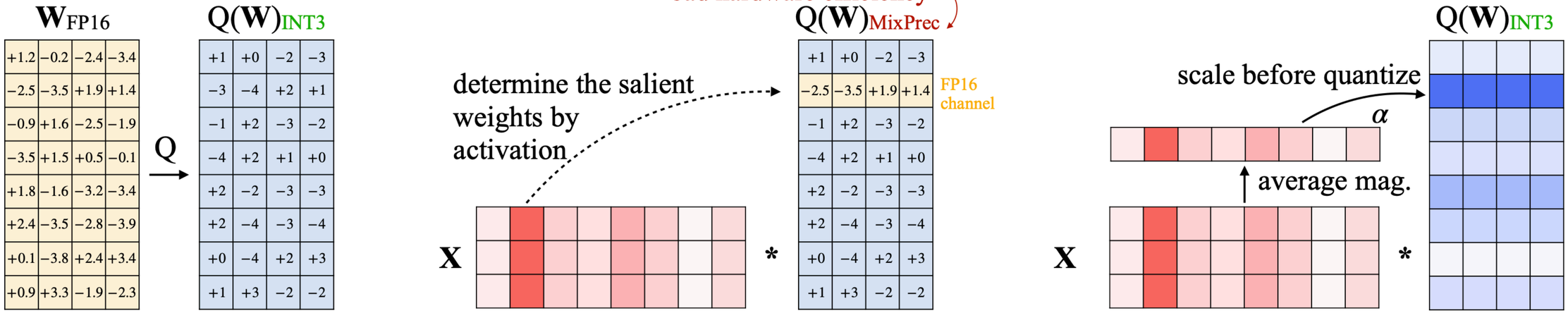
$$\mathcal{L}(\mathbf{s}) = \|Q(\mathbf{W} \cdot \mathrm{diag}(\mathbf{s}))(\mathrm{diag}(\mathbf{s})^{-1} \cdot \mathbf{X}) - \mathbf{W}\mathbf{X}\|$$

Quantization function e.g., int4

Scaling factor (per channel)

[Lin et al.]

AWQ


W FP16
Q
Q(W) INT3
bad hardware efficiency
Q(W) MixPrec
FP16 channel
determine the salient weights by activation
X
*
scale before quantize
α
average mag.
Q(W) INT3

[Lin et al.]

AWQ: Pros and cons

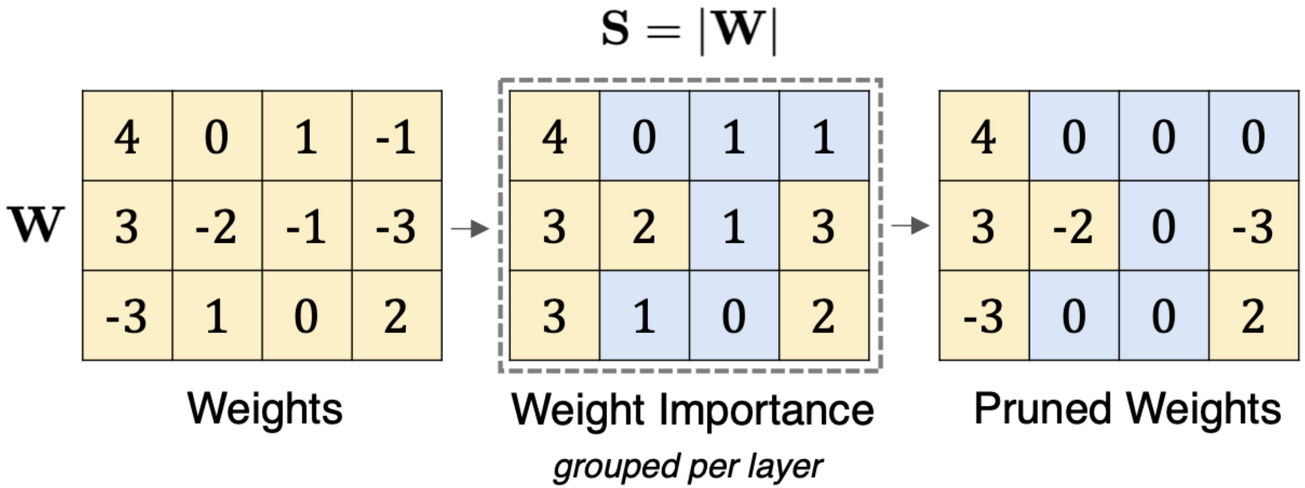
- Needs a calibration dataset
- Weight only quantization (doesn’t quantize activations)
- No mixed precision so avoids limitations of LLM.int8()
- Works well is very low precision, e.g., int4

Key considerations in quantization

- Do we need a calibration dataset?
- Is this a dynamic or static quantization strategy?
- Are only weights getting quantized or activations too?
  - Having both in int8 can leverage integer kernels to provide speedups
- Can we learn from data or do we need to make heuristic based choices?
- Does my strategy have hardware support?
  - Even int8 is not supported on all hardware
  - Some hardware does not yet support fp16
- Space savings and computational speed ups are two different things
  - Not all quantization methods will lead to a speedup, some will slow you down, e.g., LLM.int8()

Looking beyond quantization

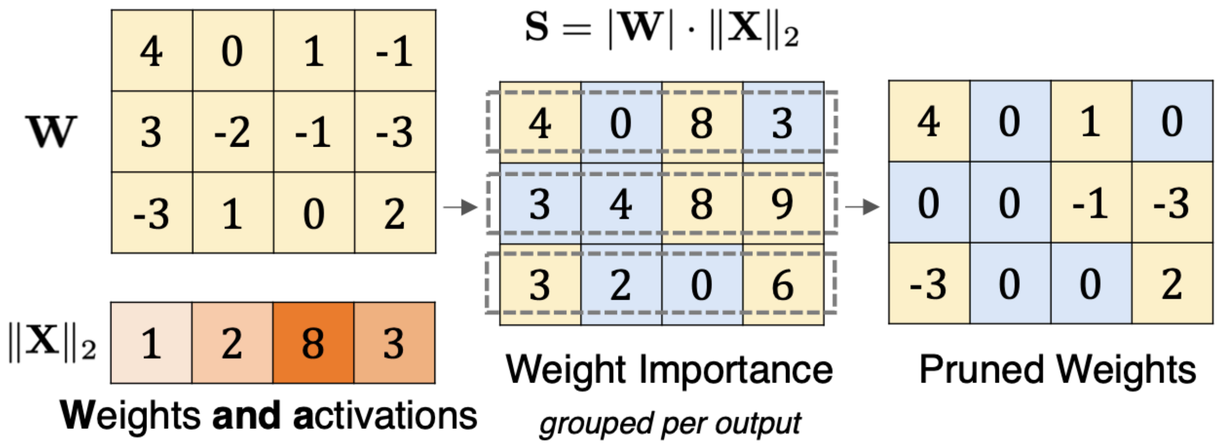
Magnitude pruning



- **Key idea:** Prune weights that are lower than a certain threshold

[Sun et al.]

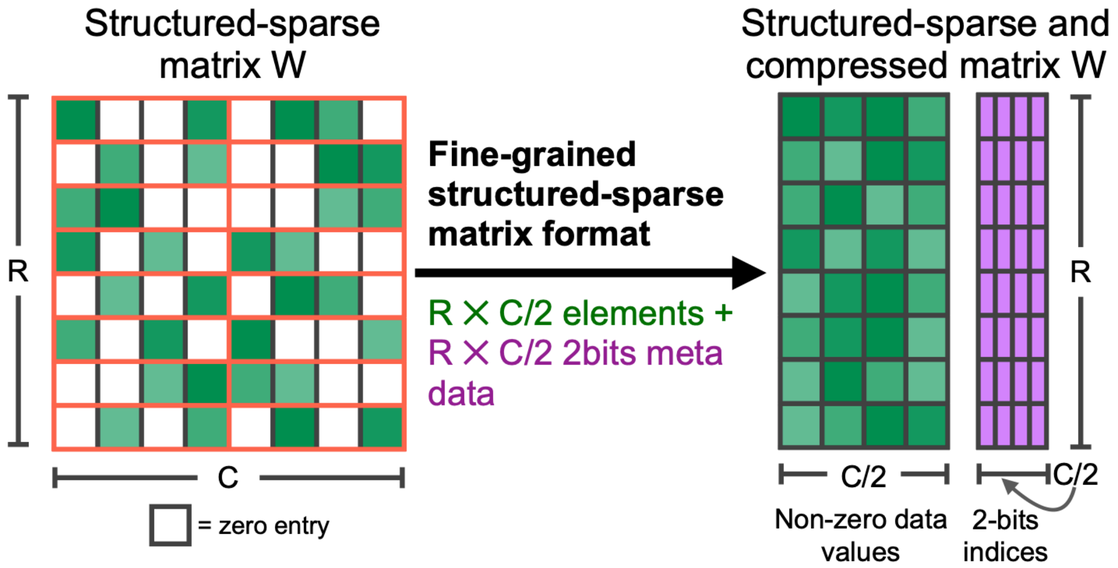
WandA (weights and activations) pruning



- **Key idea:** Prune weights that have a low importance based on the $|\mathbf{W}| \cdot ||\mathbf{X}||_2$ metric
- Similar idea to AWQ and LLM.int8(): Weight importance is determined by the activations too

[Sun et al.]

Structured pruning



- **Key idea:** Fixed ratio of weights (e.g., 2:4) is pruned between each structure
- Can leverage hardware acceleration

[Mishra et al.]

Exercise

References

- Making LLMs lighter with AutoGPTQ and transformers
- Introduction to Weight Quantization
- A Gentle Introduction to 8-bit Matrix Multiplication for transformers at scale using Hugging Face Transformers, Accelerate and bitsandbytes
- A Visual Guide to Quantization
- SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models
- GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers
- AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration
- LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale
- A Simple and Effective Pruning Approach for Large Language Models
- Accelerating Sparse Deep Neural Networks
- MIT 6.5940: TinyML and Efficient Deep Learning Computing