CS 260D: Large-scale Machine Learning

Lecture #14 & #15: Neural Network Quantization

Baharan Mirzasoleiman

UCLA Computer Science

Neural Network Quantization

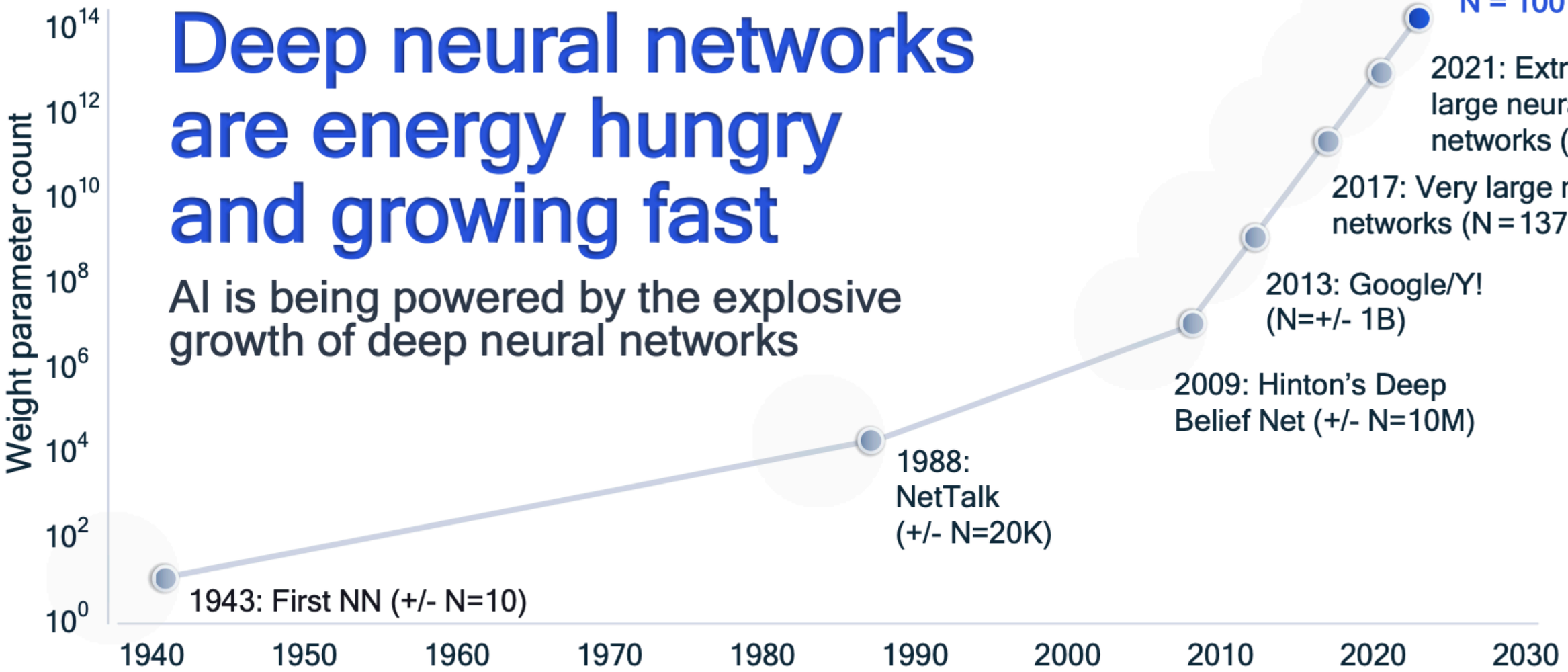
Outline

- Energy-efficient machine learning and the need for quantization
- Introduction to neural network quantization
- Simulating quantization in neural networks
- Post-training quantization (PTQ)
- Quantization-aware training (QAT)

From Marios Fournarakis

Deep neural networks are energy hungry and growing fast

AI is being powered by the explosive growth of deep neural networks



2025 | Increasingly large and complex neural networks for Natural Language Processing, Image and Video Processing

The AI power and thermal ceiling

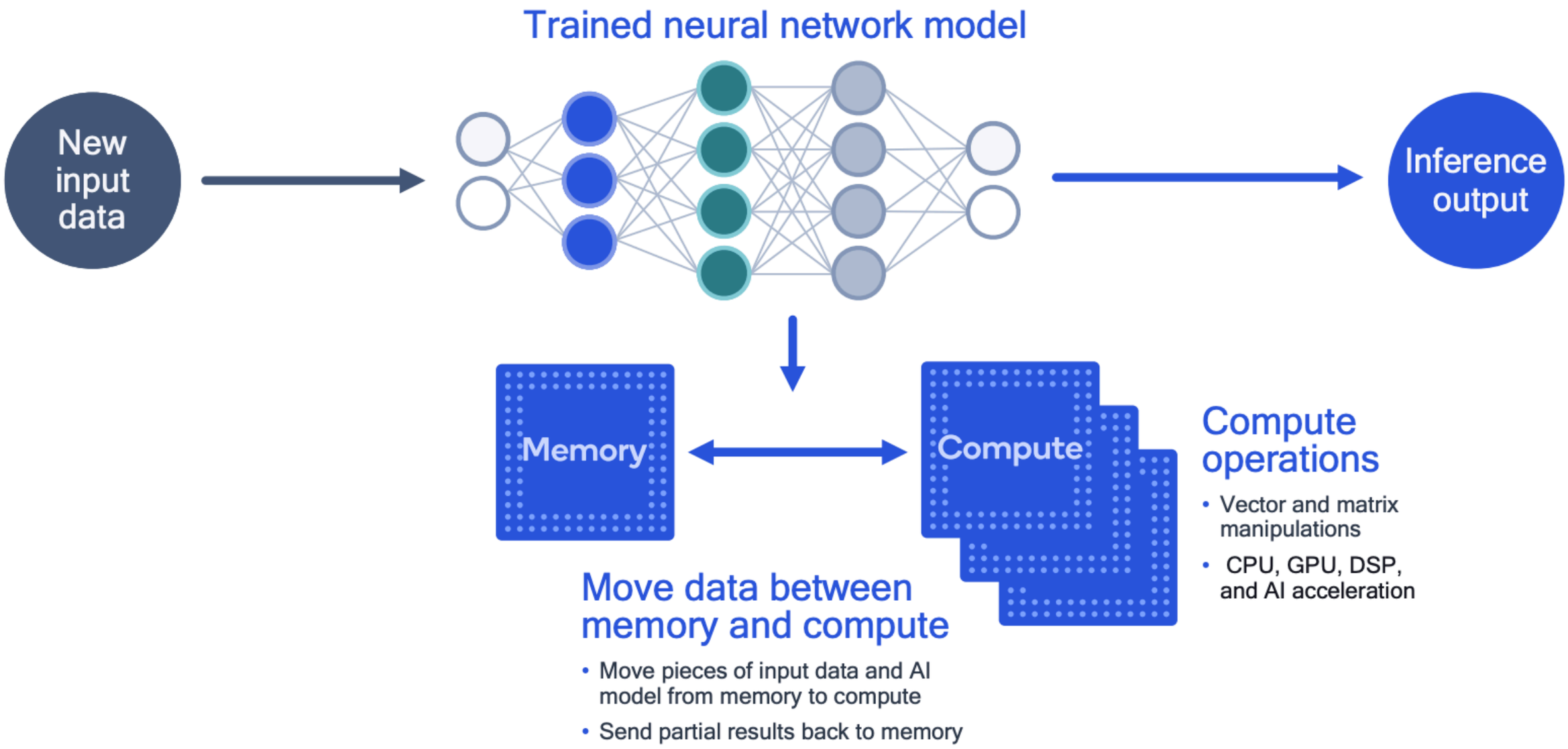
The challenge of AI workloads

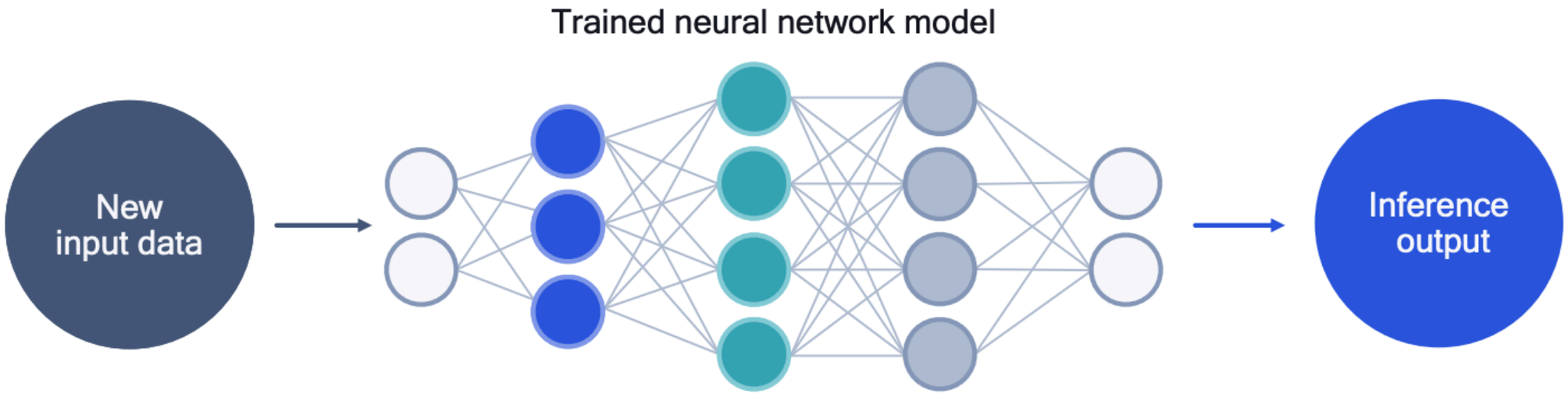
- Very compute intensive
- Complex concurrencies
- Real-time
- Always-on

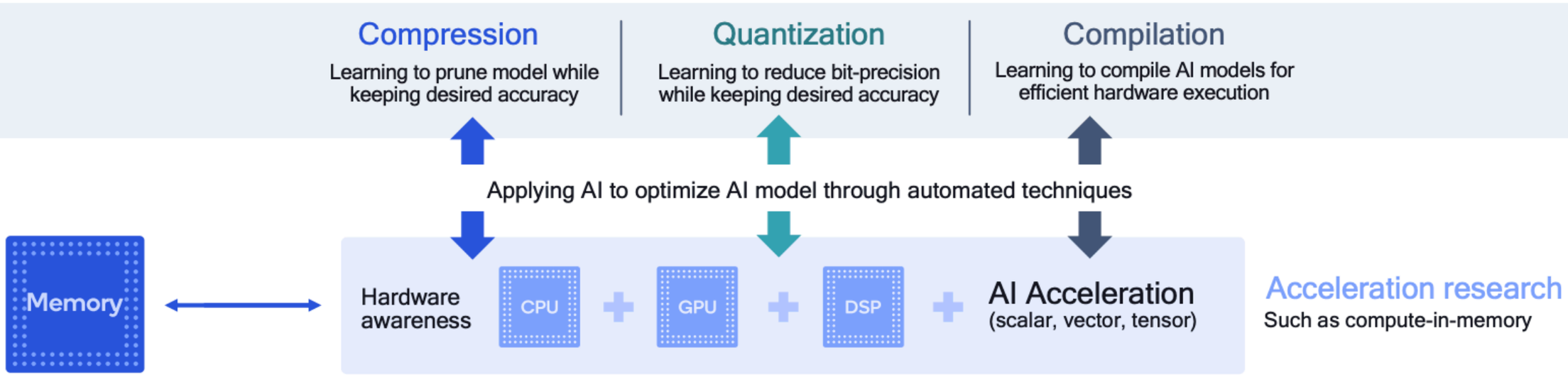
Constrained mobile environment

- Must be thermally efficient for sleek, ultra-light designs
- Requires long battery life for all-day use
- Storage/memory bandwidth limitations

Advancing AI research to increase power efficiency







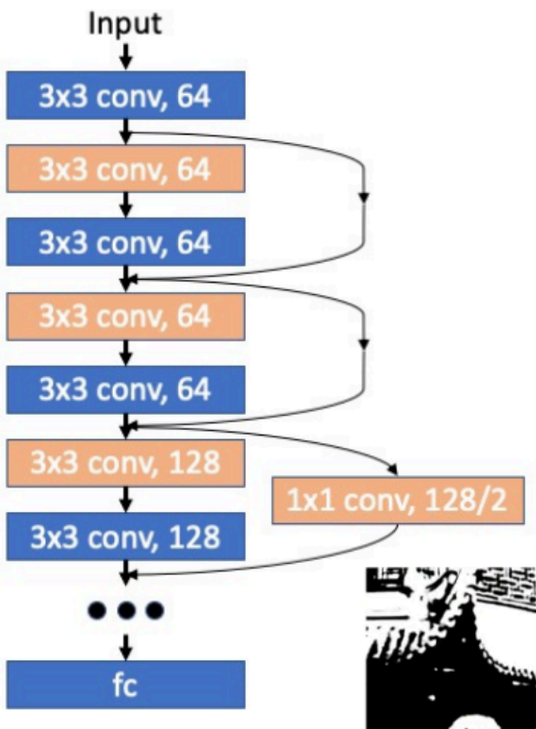
Advancing AI research to increase power efficiency

What is Neural Network Quantization?

What is neural network quantization?

For any given trained neural network:

- Store weights in low bits (INT8)
- Compute calculations in low bits



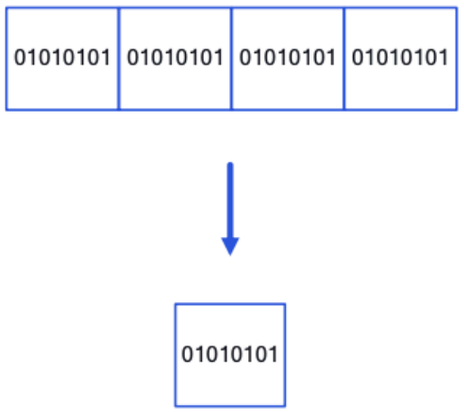
Quantization Analogy

Use fewer bits to represent each pixel in an image

Quantizing AI models offers significant benefits

Memory usage

8-bit versus 32-bit weights and activations stored in memory



Power consumption

Significant reduction in energy for both computations and memory access

Add energy (pJ)	
INT8	FP32
0.03	0.9

30X energy reduction

Mult energy (pJ)	
INT8	FP32
0.2	3.7

18.5X energy reduction

Mem access energy (pJ)	
Cache (64-bit)	
8KB	10
32KB	20
1MB	100
DRAM	1300-2600

Up to 4X energy reduction

Latency

With less memory access and simpler computations, latency can be reduced

Silicon area

Integer math or less bits require less silicon area compared to floating point math and more bits

Add area (µm²)	
INT8	FP32
36	4184

116X area reduction

Mult area (µm²)	
INT8	FP32
282	7700

27X area reduction

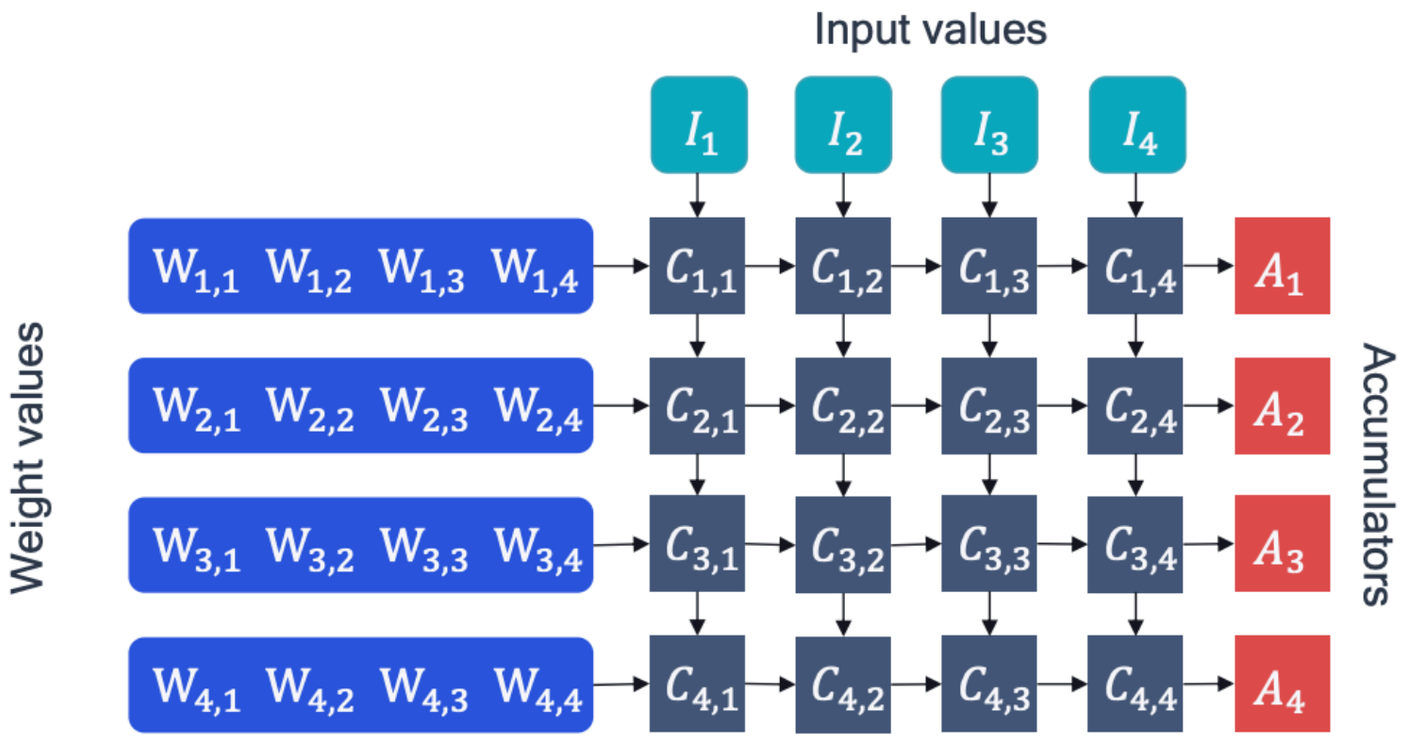
Matrix operations are the backbone of neural networks

A running example to showcase how to make these operations more efficient

$$\boldsymbol{W} = \begin{pmatrix} 0.97 & 0.64 & 0.74 & 1.00 \\ 0.58 & 0.84 & 0.84 & 0.81 \\ 0.00 & 0.18 & 0.90 & 0.28 \\ 0.57 & 0.96 & 0.80 & 0.81 \end{pmatrix} \quad \boldsymbol{X} = \begin{pmatrix} 0.41 & 0.25 & 0.73 & 0.66 \\ 0.00 & 0.41 & 0.41 & 0.57 \\ 0.42 & 0.24 & 0.71 & 1.00 \\ 0.39 & 0.82 & 0.17 & 0.35 \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{pmatrix}$$

How to most efficiently calculate $WX + b$?

A schematic MAC array for efficient computation



The array efficiently calculates the dot product between multiple vectors

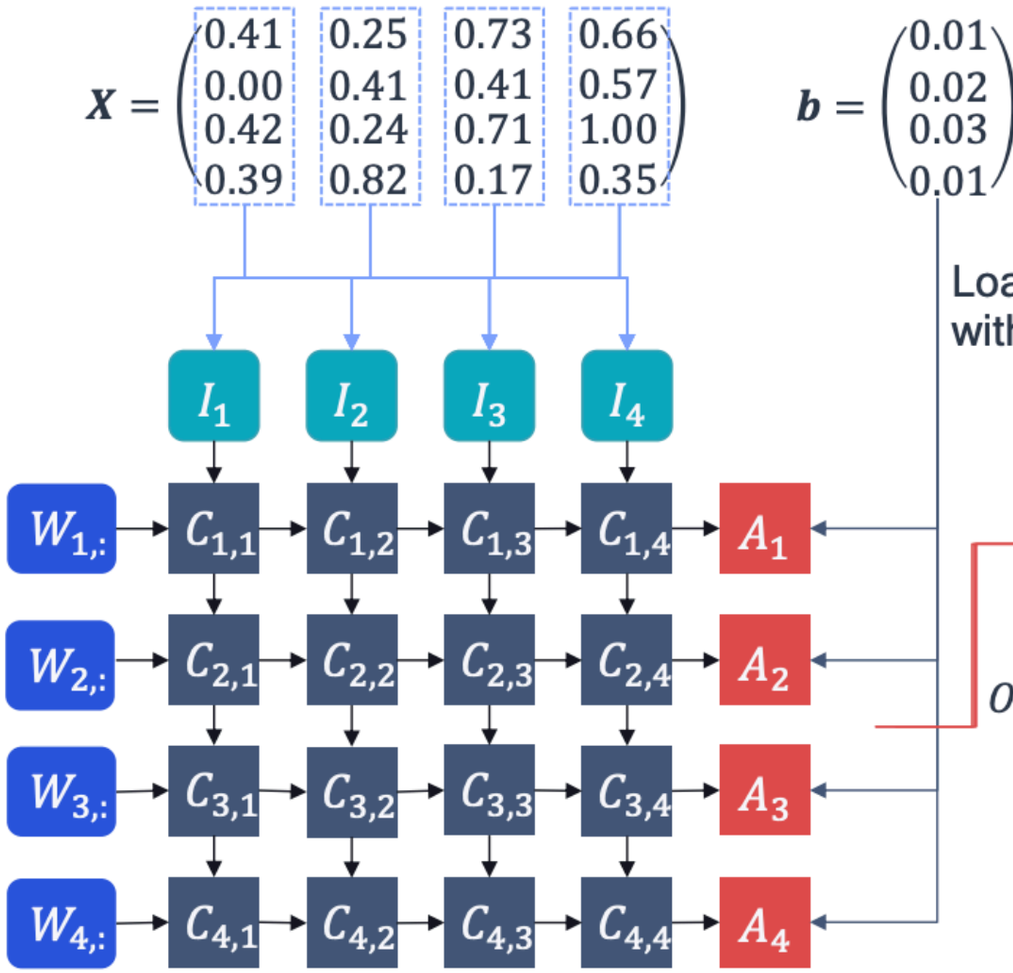
$$A_i = \sum_j C_{i,j} + \boldsymbol{b}_i$$

$$A_i = W_i \cdot I_1 + W_i \cdot I_2 + W_i \cdot I_3 + W_i \cdot I_4$$

Step-by-step matrix multiplication in MAC array

Load matrix $\boldsymbol{W}$ into MAC array

$$\boldsymbol{W} = \begin{pmatrix} 0.97 & 0.64 & 0.74 & 1.00 \\ 0.58 & 0.84 & 0.84 & 0.81 \\ 0.00 & 0.18 & 0.90 & 0.28 \\ 0.57 & 0.96 & 0.80 & 0.81 \end{pmatrix}$$



Quantization comes at a cost of lost precision

- We can approximate an FP tensor with an integer tensor multiplied by a scale-factor, s_X:

FP32 tensor → $\boldsymbol{X} \approx s_X \boldsymbol{X}_{\text{int}} = \widehat{\boldsymbol{X}}$ ← scaled quantized tensor

$$\boldsymbol{W} = \begin{pmatrix} 0.97 & 0.64 & 0.74 & 1.00 \\ 0.58 & 0.84 & 0.84 & 0.81 \\ 0.00 & 0.18 & 0.90 & 0.28 \\ 0.57 & 0.96 & 0.80 & 0.81 \end{pmatrix} \approx \frac{1}{255} \begin{pmatrix} 247 & 163 & 189 & 255 \\ 148 & 214 & 214 & 207 \\ 0 & 46 & 229 & 71 \\ 145 & 245 & 204 & 207 \end{pmatrix} = s_W \boldsymbol{W}_{\text{uint8}}$$

- Quantization is not free:

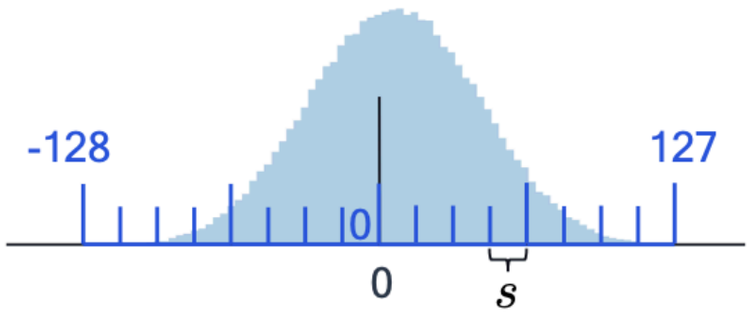
$$\boldsymbol{\epsilon} = \boldsymbol{W} - s_W \boldsymbol{W}_{\text{int}} = \frac{1}{255} \begin{pmatrix} 0.35 & 0.20 & -0.3 & 0 \\ -0.1 & 0.20 & 0.20 & -0.45 \\ 0.00 & -0.1 & -0.5 & 0.40 \\ 0.35 & -0.2 & 0 & -0.45 \end{pmatrix}$$

Different types of quantization have pros and cons

Symmetric, asymmetric, signed, and unsigned quantization

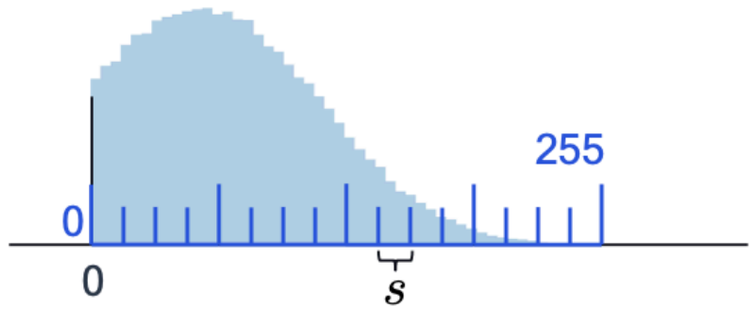
Symmetric signed

$s \cdot \mathbf{x}_{\text{int8}}$



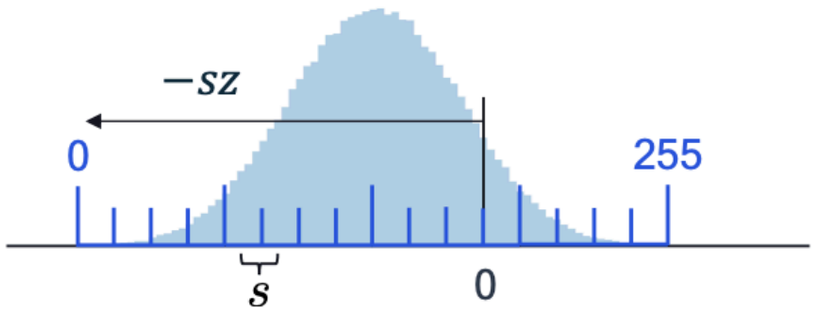
Symmetric unsigned

$s \cdot \mathbf{x}_{\text{uint8}}$



Asymmetric

$s(\mathbf{x}_{\text{uint8}} - z)$



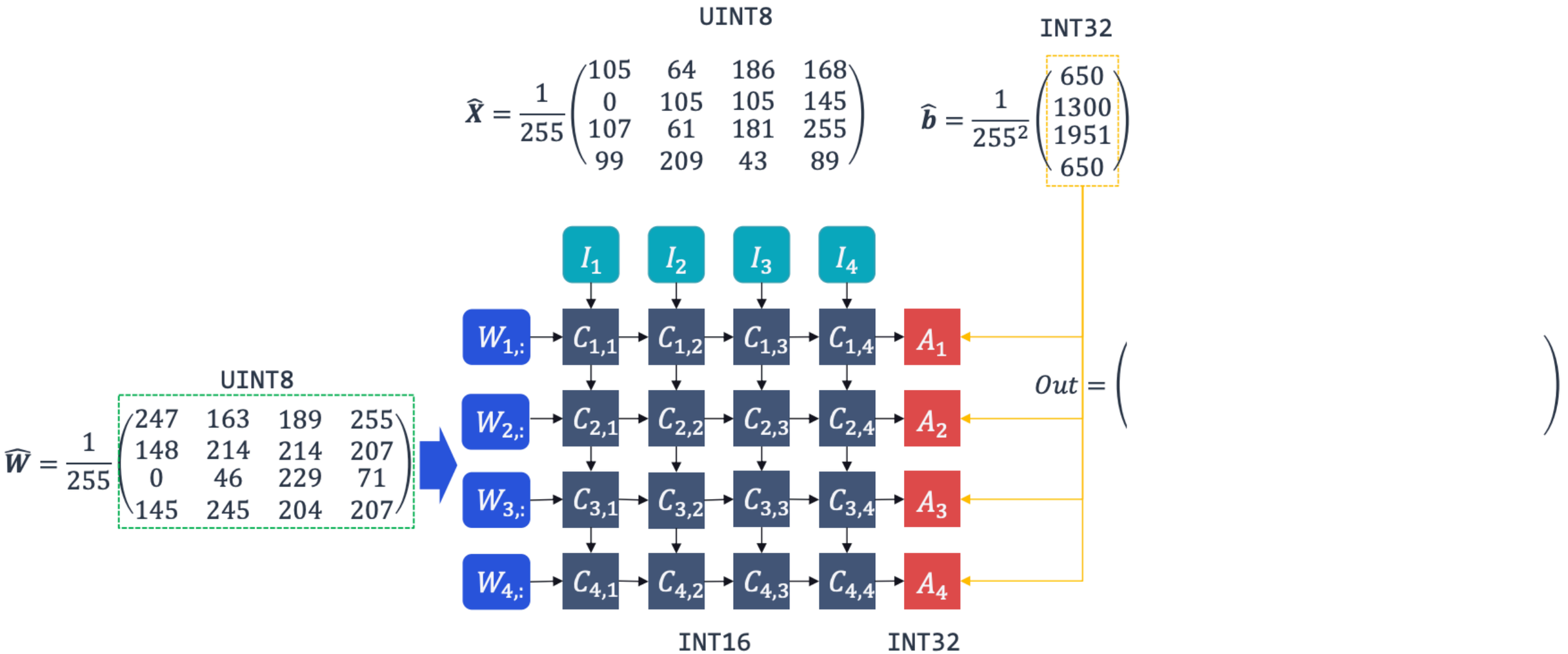
Fixed point grid

Floating point grid

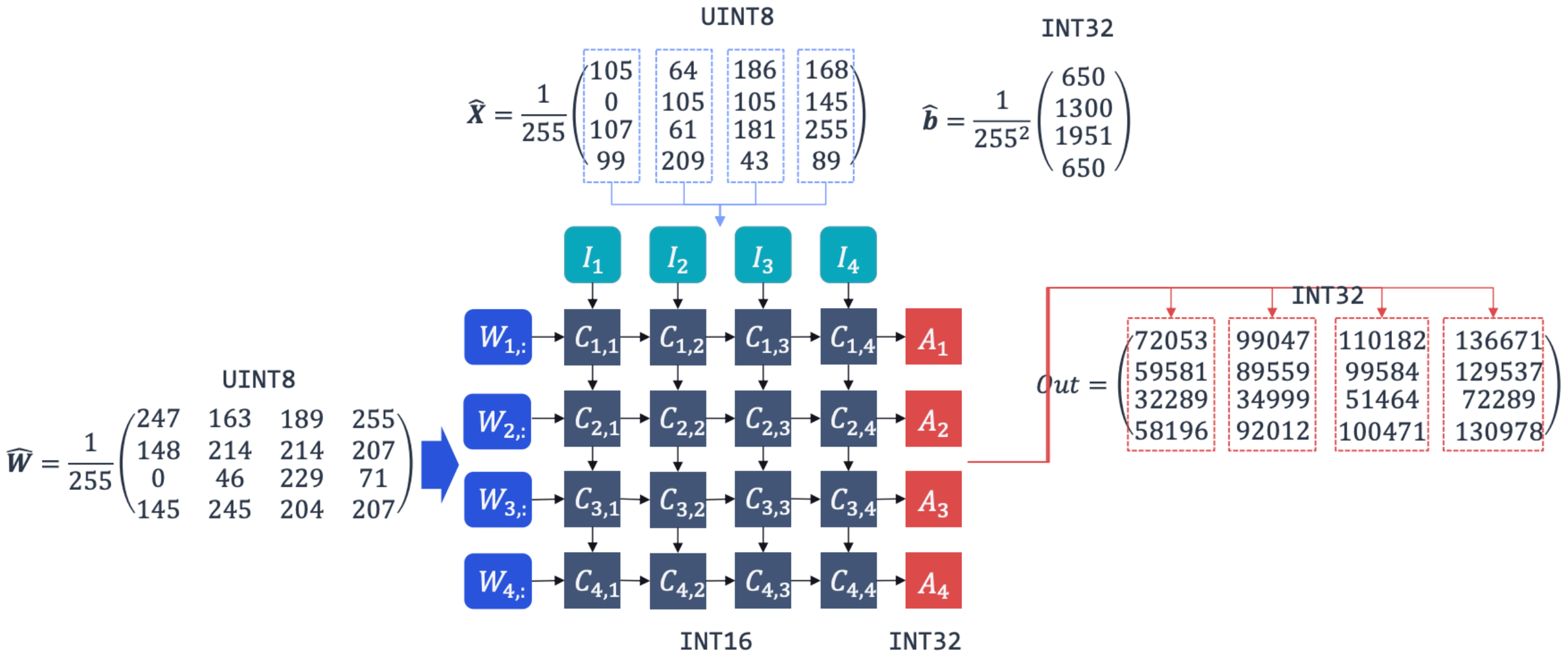
s: scale factor

z: zero-point

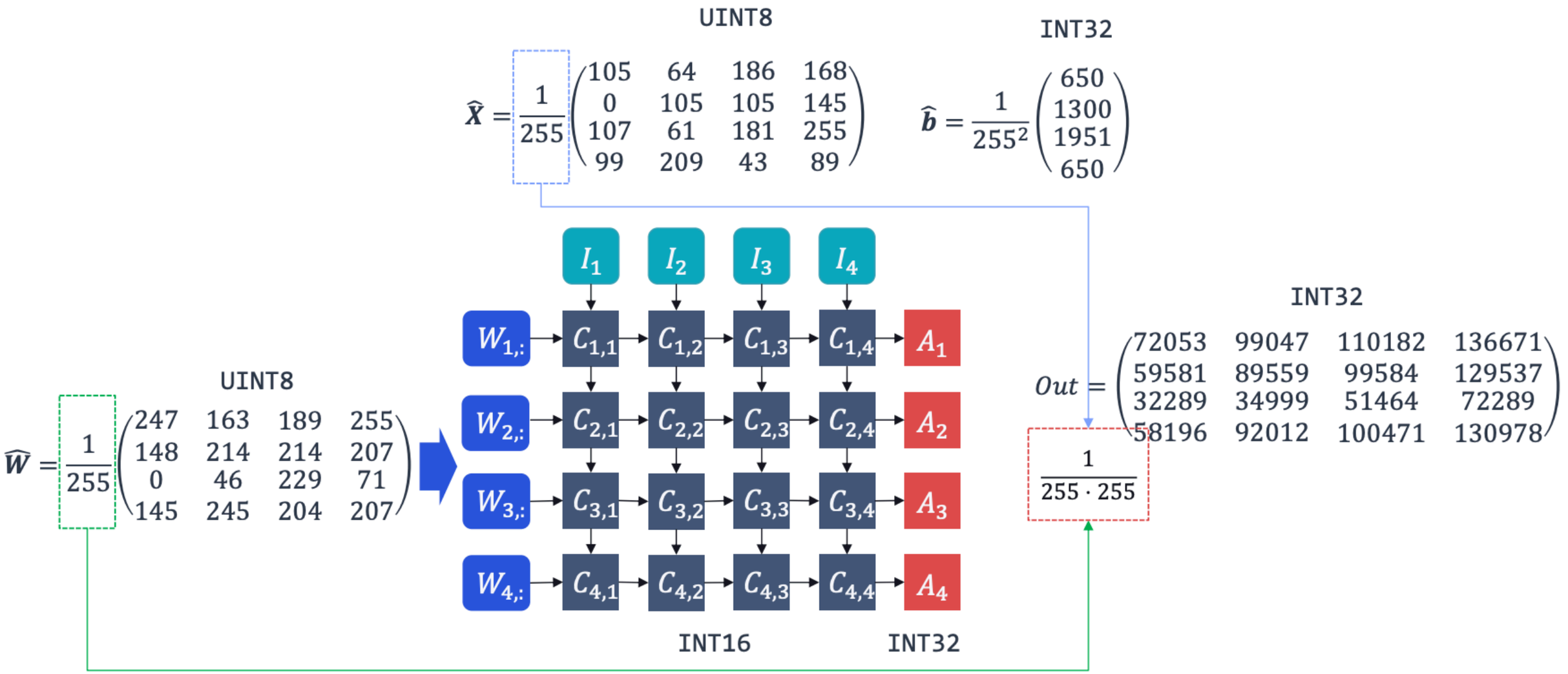
Quantized inference using symmetric quantization



Quantized inference using symmetric quantization



Quantized inference using symmetric quantization



Quantized inference using symmetric quantization

UINT8

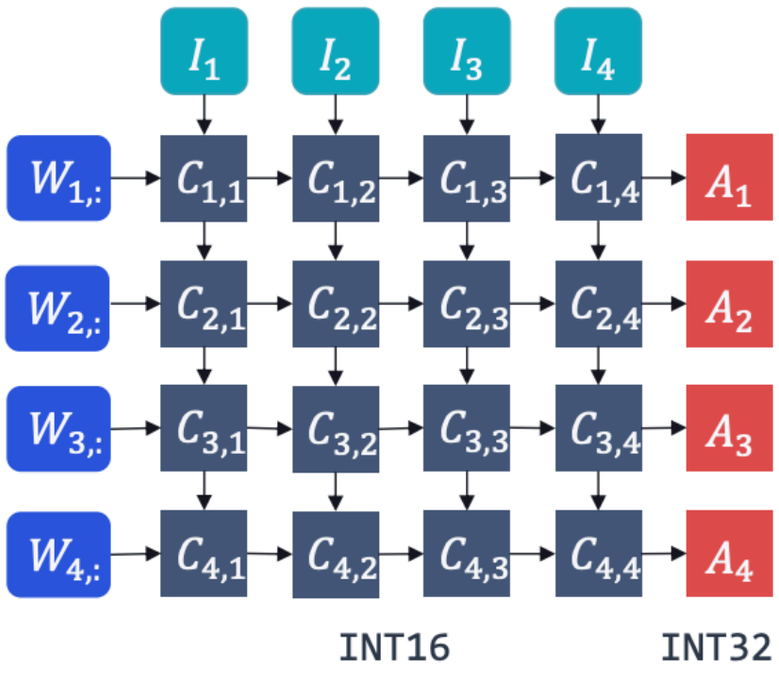
$$\widehat{X} = \frac{1}{255}\begin{pmatrix} 105 & 64 & 186 & 168 \\ 0 & 105 & 105 & 145 \\ 107 & 61 & 181 & 255 \\ 99 & 209 & 43 & 89 \end{pmatrix}$$

INT32

$$\widehat{b} = \frac{1}{255^2}\begin{pmatrix} 650 \\ 1300 \\ 1951 \\ 650 \end{pmatrix}$$

UINT8

$$\widehat{W} = \frac{1}{255}\begin{pmatrix} 247 & 163 & 189 & 255 \\ 148 & 214 & 214 & 207 \\ 0 & 46 & 229 & 71 \\ 145 & 245 & 204 & 207 \end{pmatrix}$$



INT32

$$Out = \begin{pmatrix} 72053 & 99047 & 110182 & 136671 \\ 59581 & 89559 & 99584 & 129537 \\ 32289 & 34999 & 51464 & 72289 \\ 58196 & 92012 & 100471 & 130978 \end{pmatrix}$$

$\frac{1}{255 \cdot 255}$ → UINT8 Activation quantization

$$\widehat{Out} = \frac{1}{136671 \cdot 255}\begin{pmatrix} 134 & 185 & 206 & 255 \\ 111 & 167 & 186 & 242 \\ 60 & 65 & 96 & 134 \\ 109 & 172 & 187 & 244 \end{pmatrix}$$

What type of quantization should you use?

$\boldsymbol{W}$: weight matrix

$\boldsymbol{X}$: input of a layer

Symmetric quantization

$$\boldsymbol{WX} \approx s_W(\boldsymbol{W}_{\text{int}})\, s_X(\boldsymbol{X}_{\text{int}})$$
$$= s_W s_X(\boldsymbol{W}_{\text{int}}\, \boldsymbol{X}_{\text{int}})$$

Same calculation

Asymmetric quantization

$$\boldsymbol{WX} \approx s_W(\boldsymbol{W}_{\text{int}} - z_W)\, s_X(X_{\text{int}} - z_X)$$
$$= s_W s_X(\boldsymbol{W}_{\text{int}}\, \boldsymbol{X}_{\text{int}}) + s_W s_X z_X \boldsymbol{W}_{\text{int}} + s_W z_W s_X z_X + s_W s_X z_W \boldsymbol{X}_{\text{int}}$$

Precompute, add to layer bias

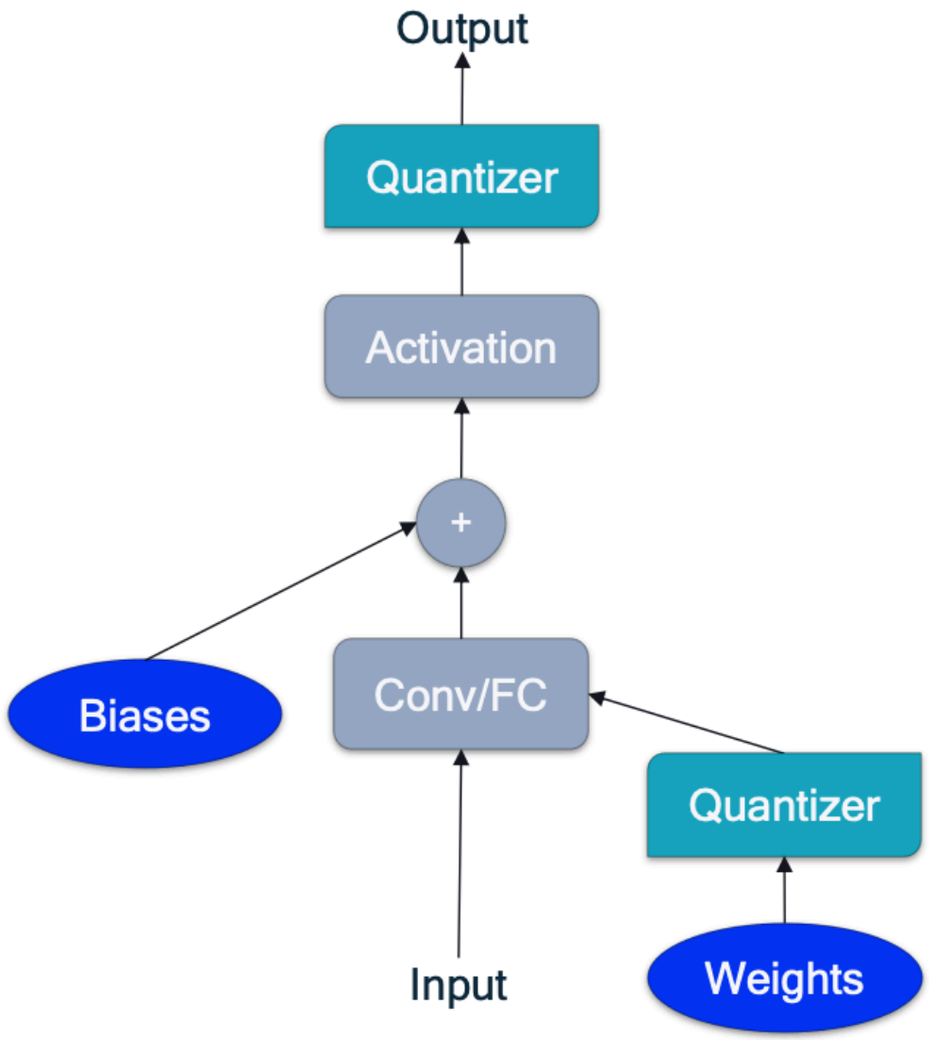
Data-dependent overhead

Asymmetric weight quantization is equivalent to adding an input channel

Symmetric weights and asymmetric activations more hardware efficient

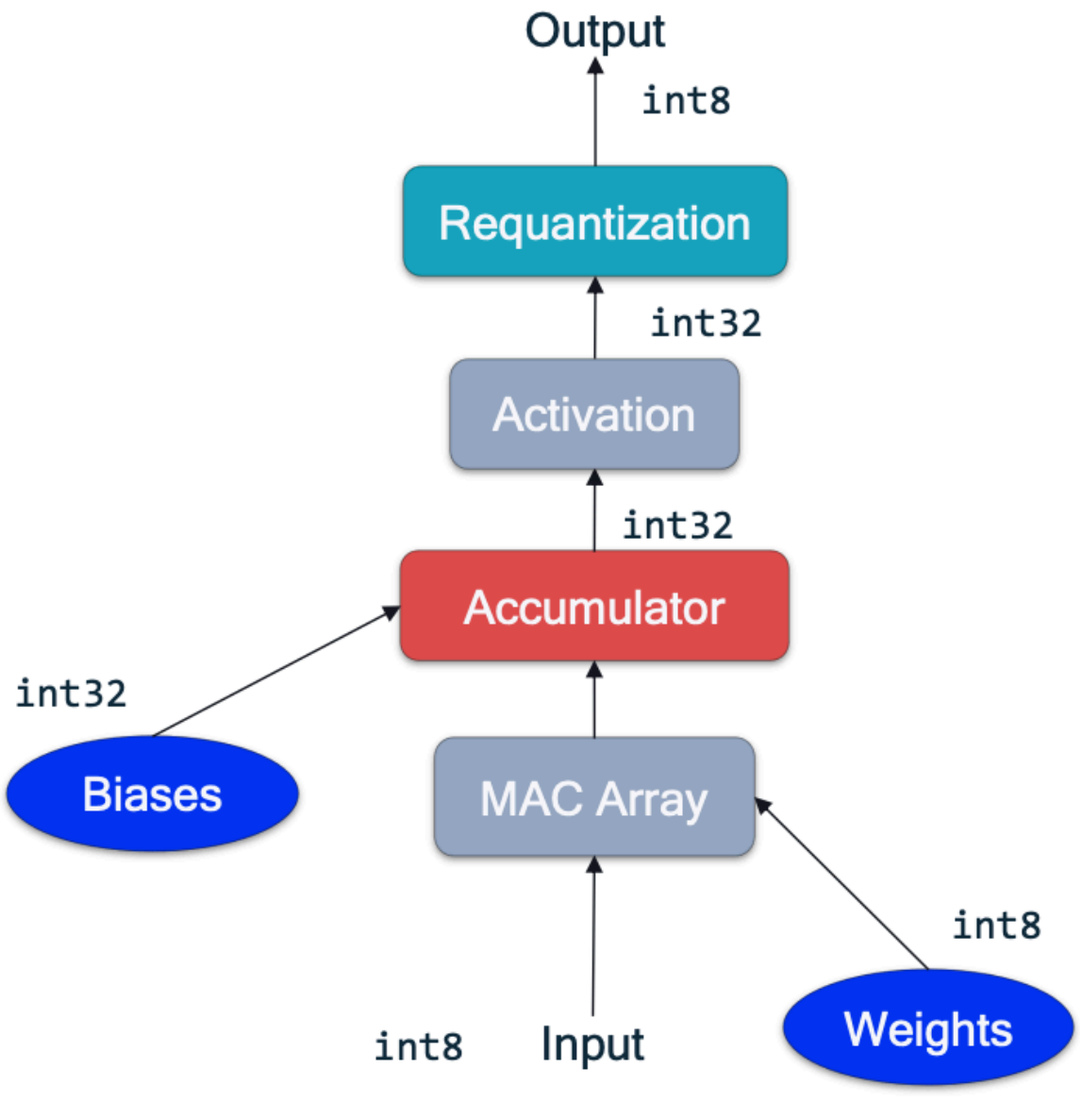
Simulating Quantization

Why simulate quantization?

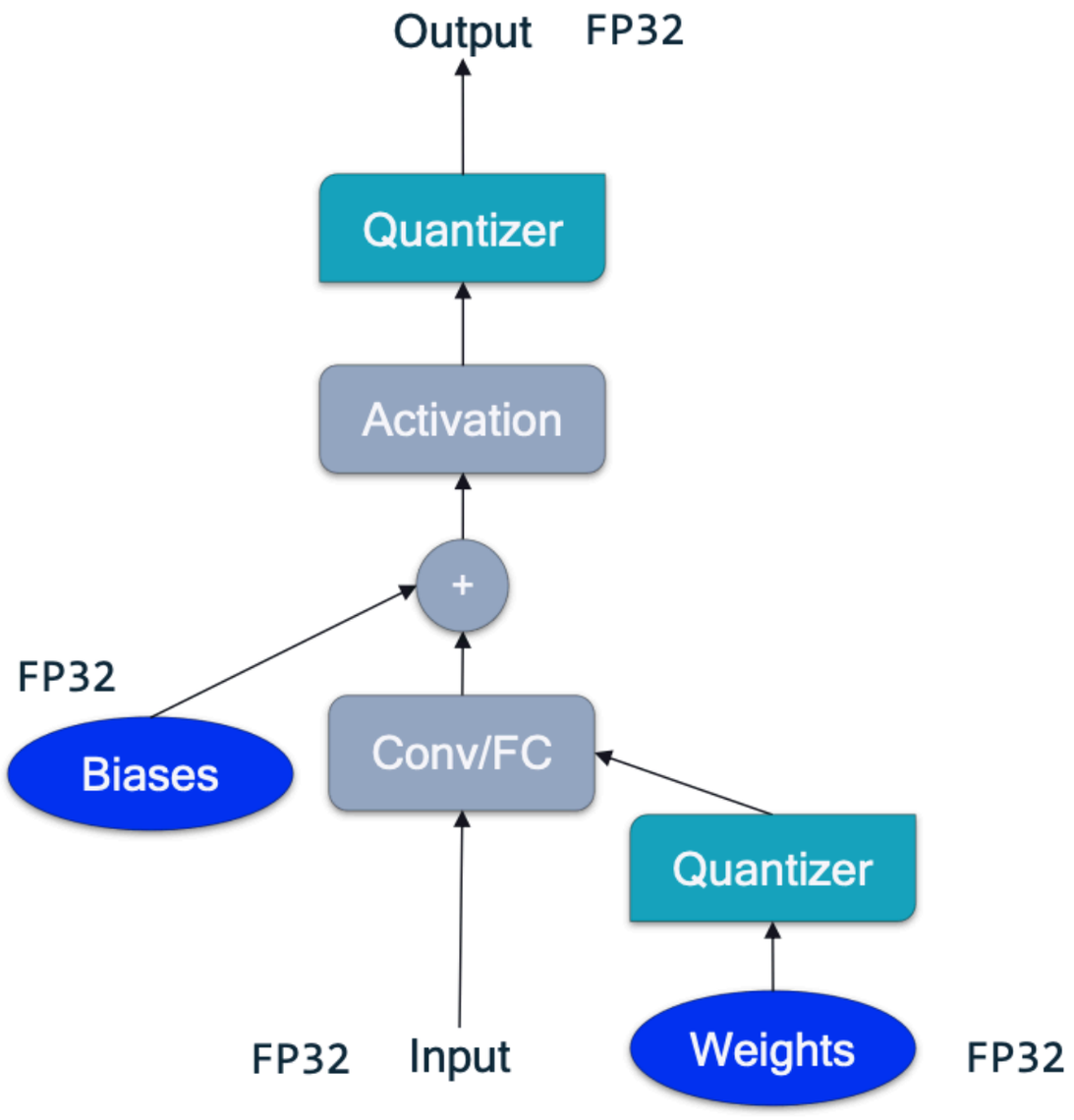


- We simulate fixed-point operations with floating-point numbers using general purpose hardware (e.g. CPU, GPU)
- This simulation is achieved by introducing simulated quantization operations (quantizers) to the compute graph.
- Quantization simulation benefits:
  - Enables GPUs acceleration
  - No need for dedicated kernels
  - Test various quantization option and bit-widths

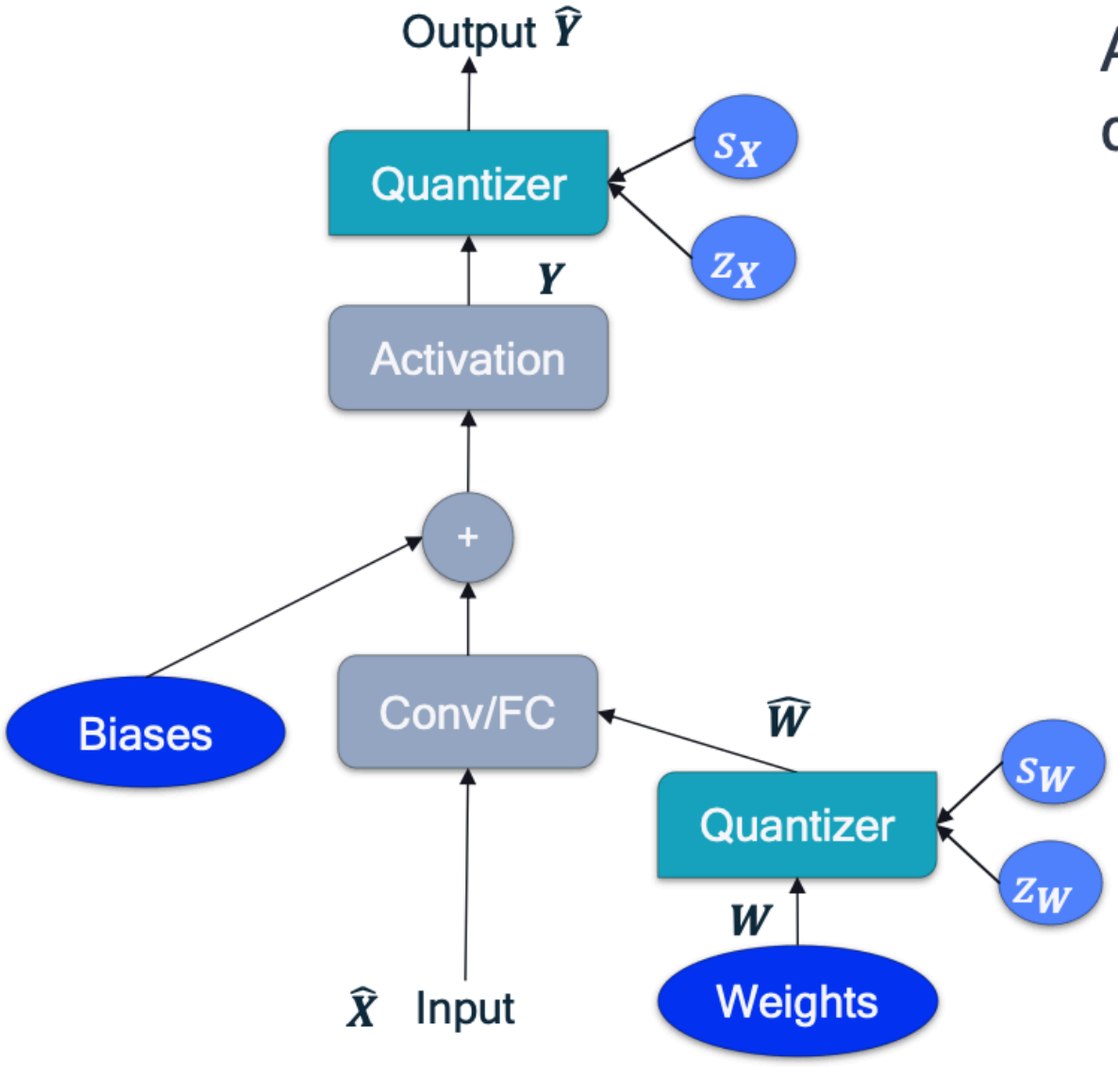
On-device fixed-point inference



Simulated quantized inference



What operations do the quantizer perform?



Assuming asymmetric quantization the quantization operation applied to input tensor $\boldsymbol{X}$:

$$\boldsymbol{X}_{\text{int}} = \text{clip}\left(\text{round}\left(\frac{\boldsymbol{X}}{s}\right) + z, \text{min} = 0, \text{max} = 2^b - 1\right)$$

$$\hat{\boldsymbol{X}} = s\,(\boldsymbol{X}_{\text{int}} - z)$$

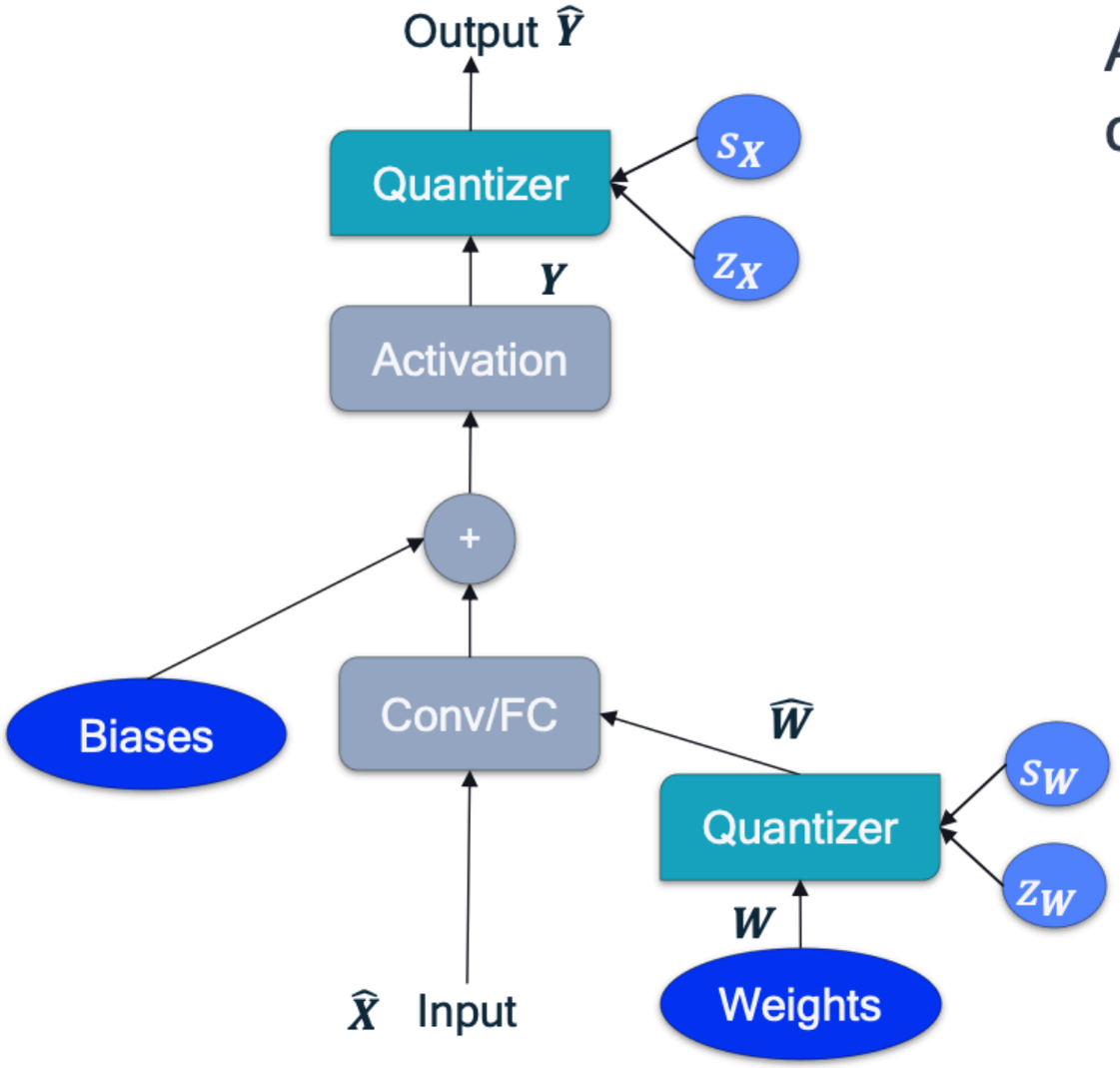
Example using $b = 4$:

$$X = \begin{pmatrix} 0.41 & 0.0 \\ 0.8 & -0.5 \end{pmatrix}$$

$$s = \frac{1}{15} = 0.067$$

$$z = \text{round}\left(\frac{0.5}{0.067}\right) = 8$$

What operations do the quantizer perform?



Assuming asymmetric quantization the quantization operation applied to input tensor $\boldsymbol{X}$:

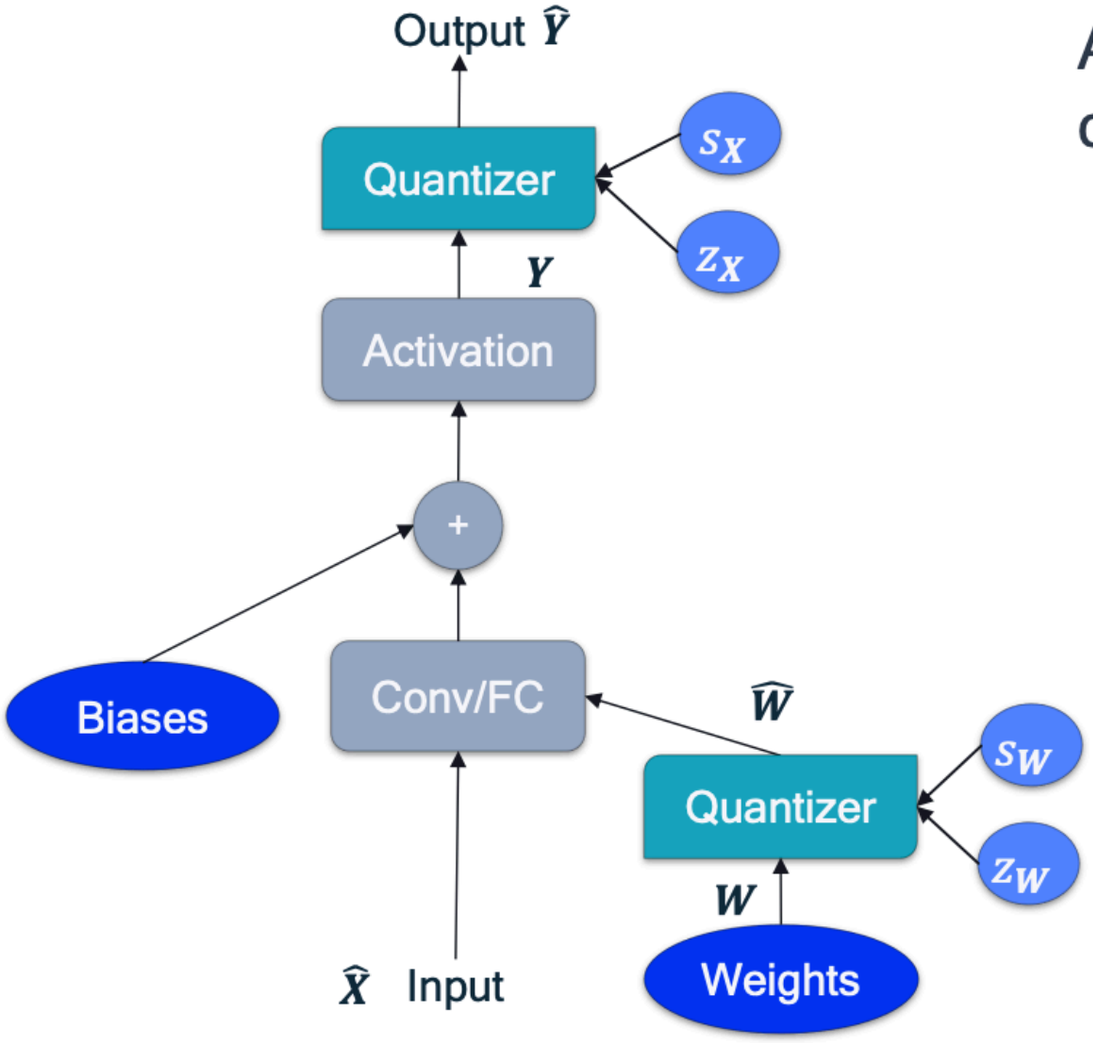
$$\boldsymbol{X}_{\text{int}} = \text{clip}\left(\text{round}\left(\frac{\boldsymbol{X}}{s}\right) + z, \min = 0, \max = 2^b - 1\right)$$

$$\hat{\boldsymbol{X}} = s\,(\boldsymbol{X}_{\text{int}} - z) \qquad \text{round}\left(\frac{X}{s}\right) + z = \begin{pmatrix} 14 & 8 \\ 20 & 0 \end{pmatrix}$$

Example using $b = 4$: $\quad s = 0.067 \quad z = 8$

$$\frac{X}{s} = \begin{pmatrix} 6.15 & 0.0 \\ 12 & -7.5 \end{pmatrix}$$

What operations do the quantizer perform?



Assuming asymmetric quantization the quantization operation applied to input tensor $\boldsymbol{X}$:

$$\boldsymbol{X}_{\text{int}} = \text{clip}\left(\text{round}\left(\frac{\boldsymbol{X}}{s}\right) + z, \text{min} = 0, \text{max} = 2^b - 1\right)$$

$$\hat{\boldsymbol{X}} = s\,(\boldsymbol{X}_{\text{int}} - z) \qquad \text{round}\left(\frac{X}{s}\right) + z = \begin{pmatrix} 14 & 8 \\ 20 & 0 \end{pmatrix}$$

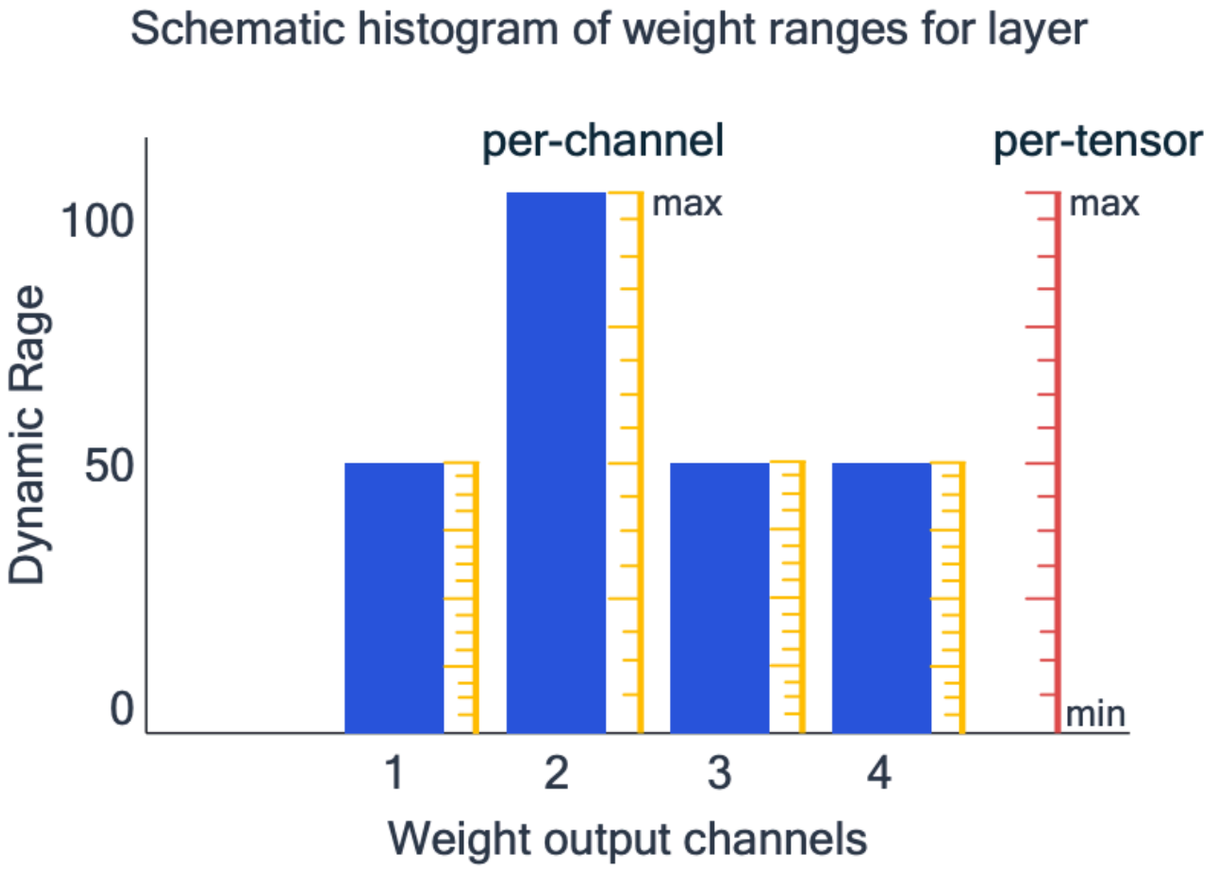
Example using $b = 4$: $\quad s = 0.067 \quad z = 8$

$$\text{round}\left(\frac{X}{s}\right) + z = \begin{pmatrix} 14 & 8 \\ 20 & 0 \end{pmatrix} \xrightarrow{\text{clip}} \begin{pmatrix} 14 & 8 \\ \mathbf{15} & 0 \end{pmatrix}$$

de-quantize ↓

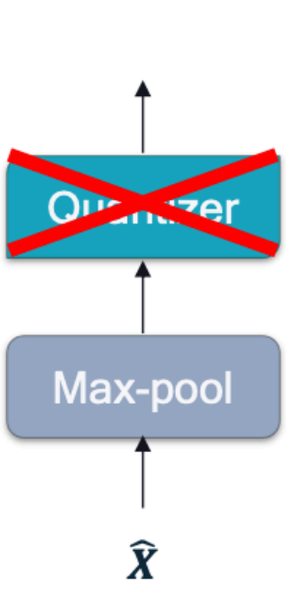
$$X = \begin{pmatrix} 0.41 & 0.0 \\ 0.8 & -0.5 \end{pmatrix} \qquad \hat{X} = \begin{pmatrix} 0.4 & 0.0 \\ 0.47 & -0.53 \end{pmatrix}$$

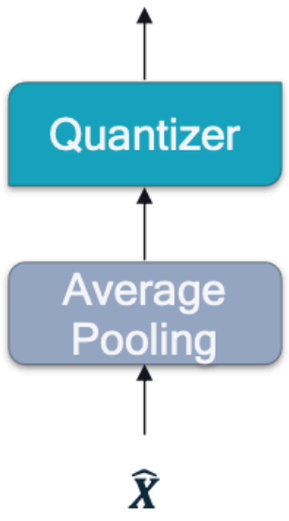
Per-channel vs Per-tensor quantization of weights



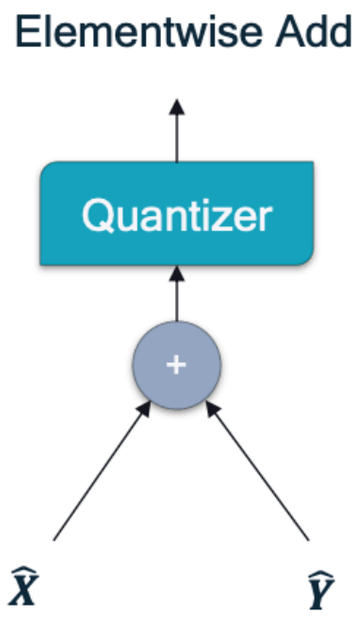
- Per-tensor quantization most supported by fixed-point accelerators
- Per-channel quantization better utilizes the quantization grid
- Per-channel quantization increasingly popular for weights
- Check for HW support

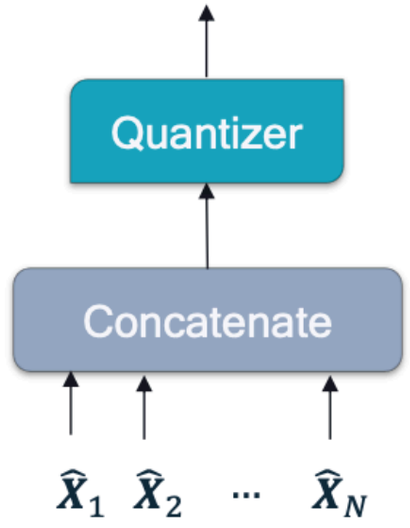
How to simulate quantization in common DL layers





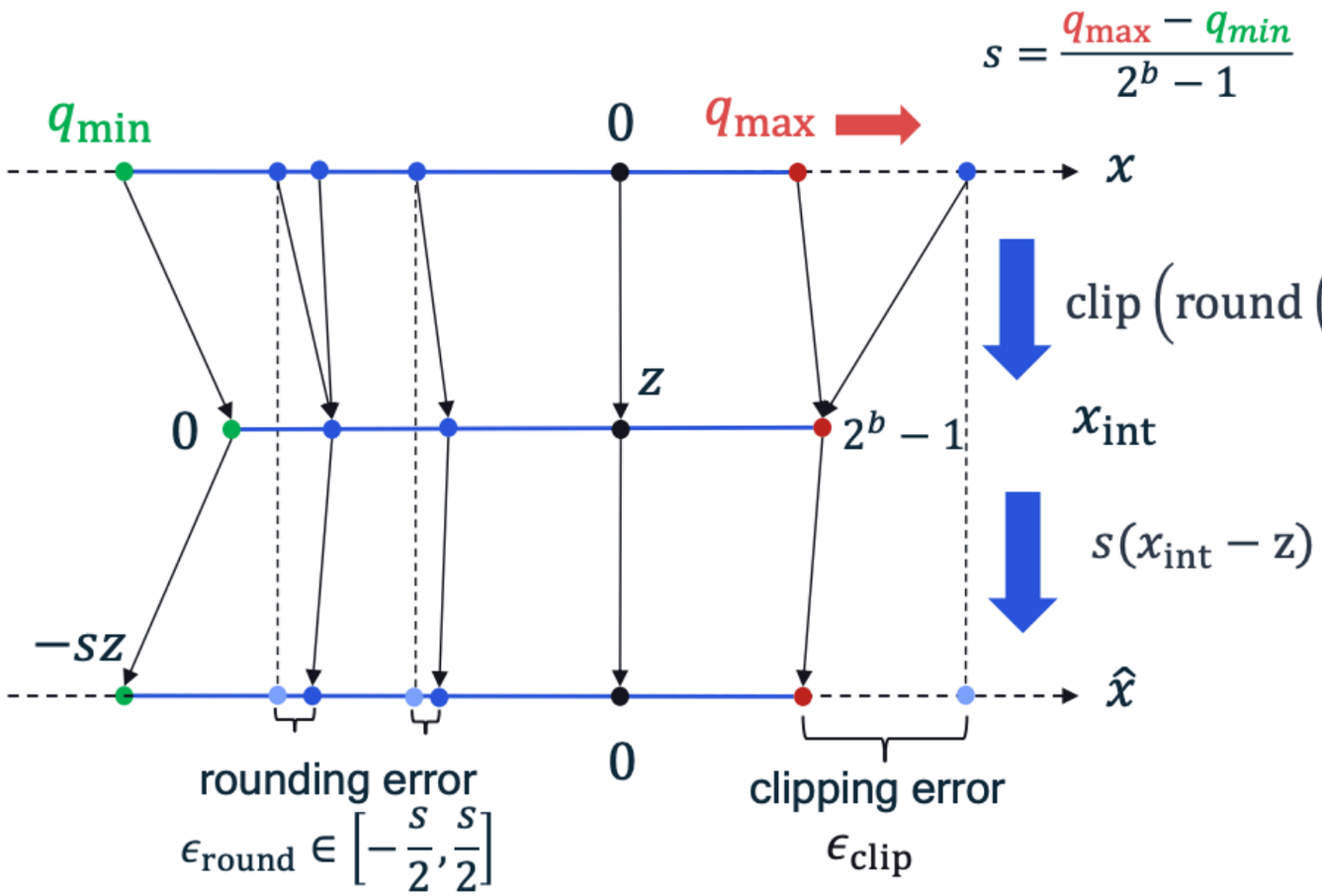
We can tie input and output quantizers



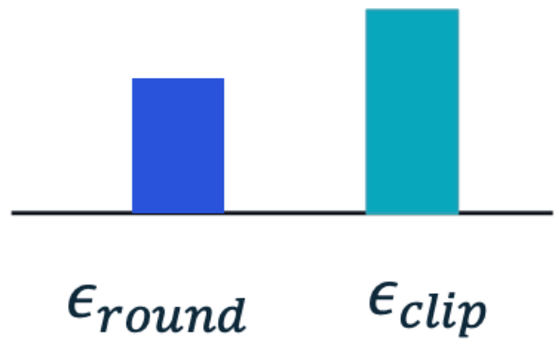


Choosing the quantization parameters

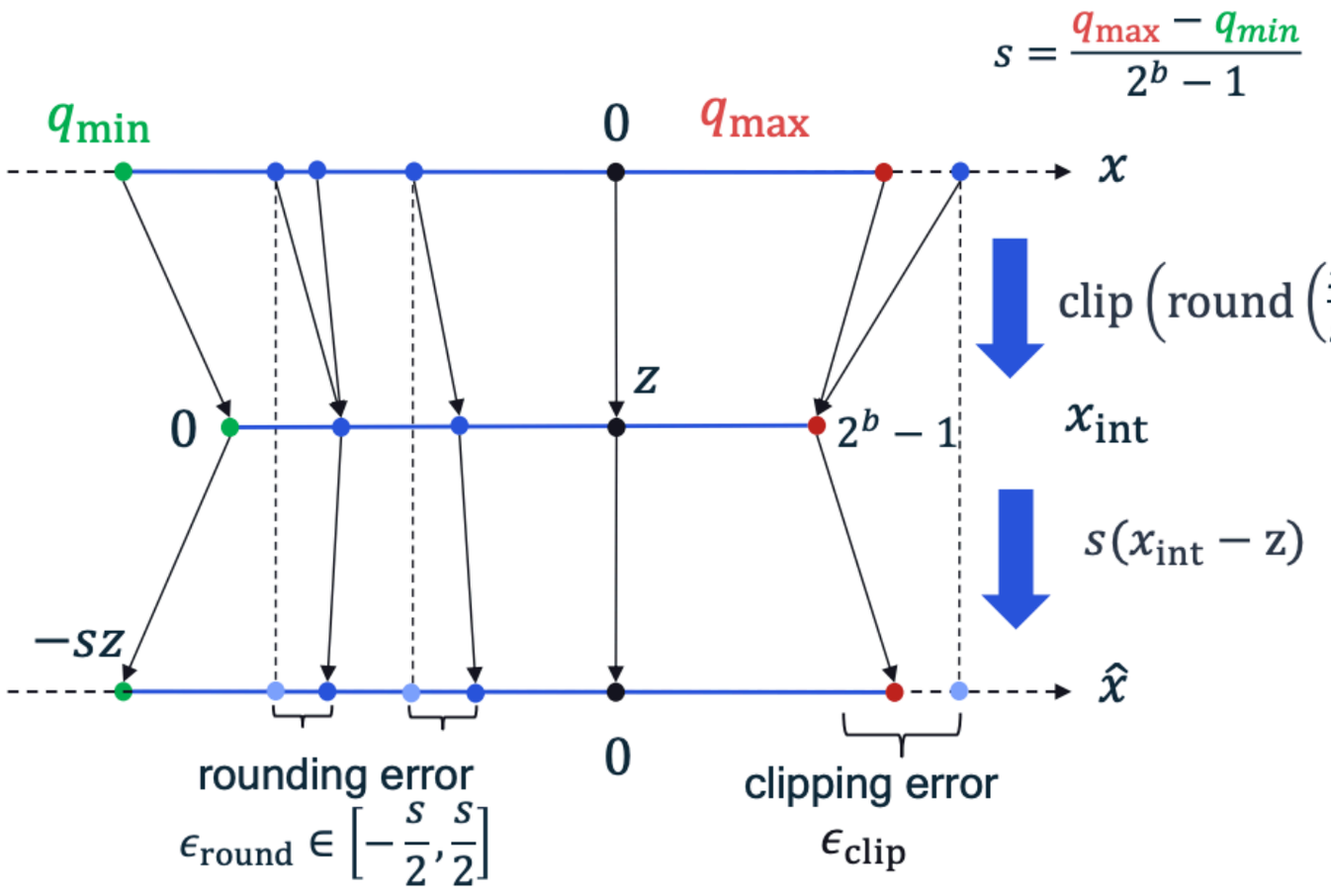
Sources of quantization error



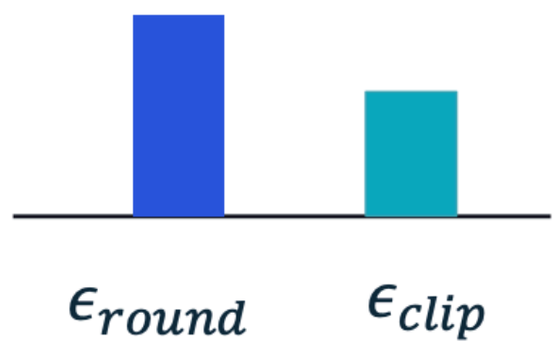
$$\epsilon_{quant} = \sum_{data} \epsilon_{round} + \epsilon_{clip}$$



Sources of quantization error



$$\epsilon_{quant} = \sum_{data} \epsilon_{round} + \epsilon_{clip}$$



Quantization range setting methods

- Min-max range:

$$q_{\min} = \min \boldsymbol{X}$$
$$q_{\max} = \max \boldsymbol{X}$$

- Optimization-based methods:

$$\operatorname{argmin}_{q_{\min}, q_{\max}} \ell\left(\boldsymbol{X}, \widehat{\boldsymbol{X}}(q_{\min}, q_{\max})\right)$$

MSE Cross-entropy

- Batch-Norm Based [1]:

$$q_{\min} = \min\left(\boldsymbol{\beta} - \alpha\boldsymbol{\gamma}\right)$$
$$q_{\max} = \max\left(\boldsymbol{\beta} + \alpha\boldsymbol{\gamma}\right)$$

$$\text{BatchNorm}\left(\boldsymbol{z}_k\right) = \boldsymbol{\gamma}_k \frac{\boldsymbol{z}_k - \boldsymbol{\mu}_k}{\sqrt{\boldsymbol{\sigma}_k + \epsilon}} + \boldsymbol{\beta}_k$$

[1] Nagel et al, 2019, Data-Free Quantization Through Weight Equalization and Bias Correction

Quantization setting methods ablation study

Model (FP32 Accuracy)	ResNet18 (69.68)		MobileNetV2 (71.72)	
Bit-width	A8	A6	A8	A6
Min-Max	69.60	68.19	70.96	64.58
MSE	69.59	67.84	71.35	67.55
MSE & X-entropy	69.60	68.91	71.36	68.85
BN ($\alpha = 6$)	69.54	68.73	71.32	71.32

Average ImageNet validation accuracy (%) over 5 seeds

Post-Training Quantization (PTQ)

- ✓ Takes a pre-trained network and converts it to a fixed-point network without access to the training pipeline
- ✓ Data-free or small calibration set needed
- ✓ Use though single API call
- ✗ Lower accuracy at lower bit-widths

Quantization-Aware Training (QAT)

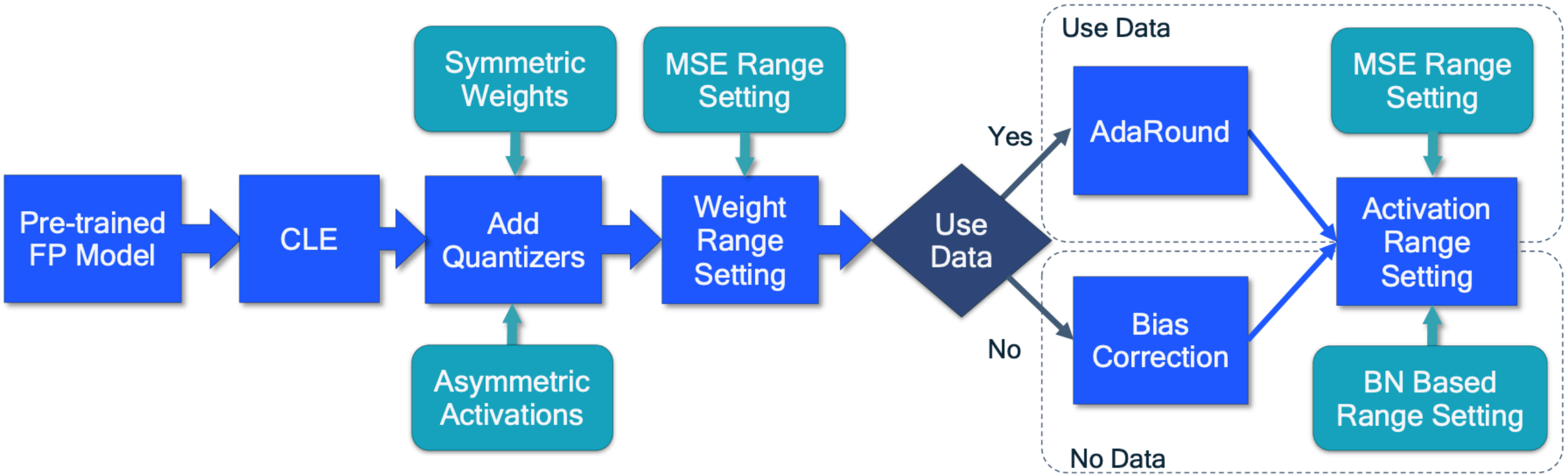
- ✗ Requires access to training pipeline and labelled data
- ✗ Longer training times
- ✗ Hyper-parameter tuning
- ✓ Achieves higher accuracy

Source sample text

What algorithm to choose to improve accuracy?

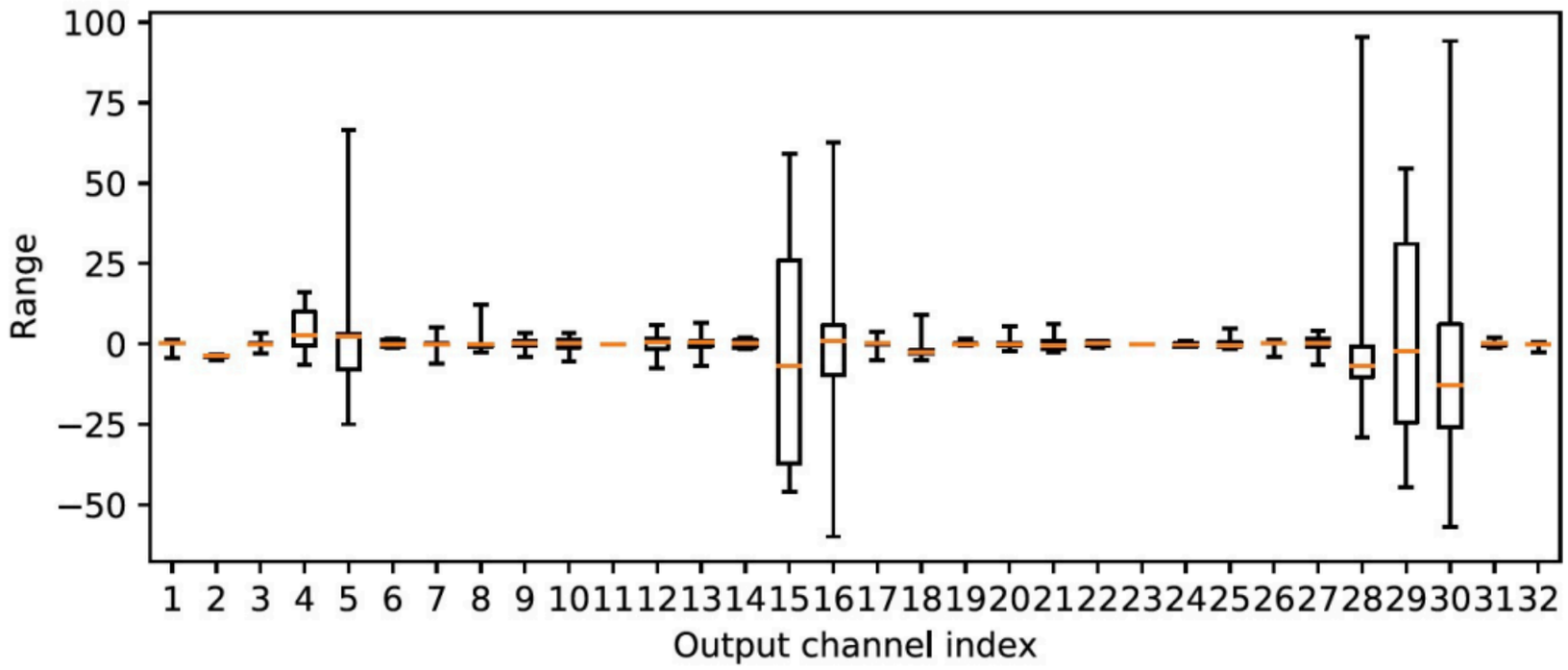
Post training quantization

Post-training quantization pipeline



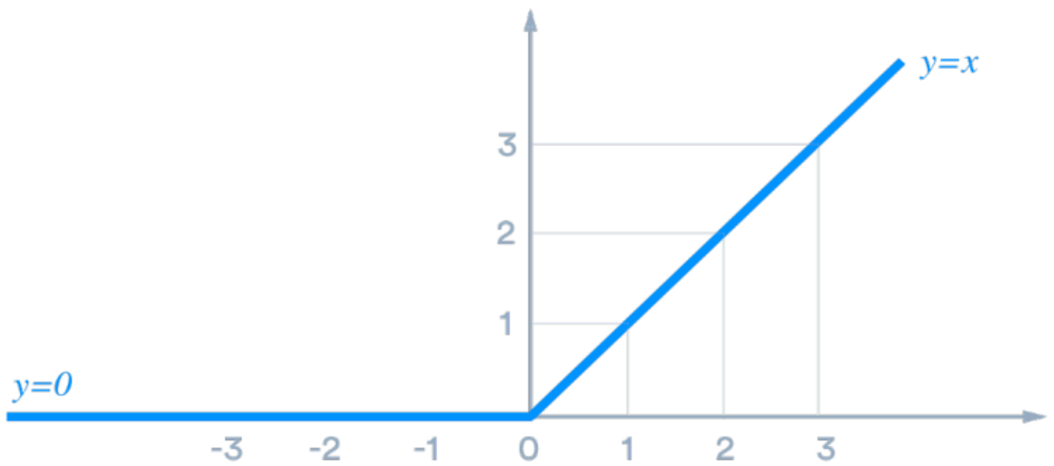
Cross-Layer Equalization

Imbalanced weights is a common problem in practice



Distributions of weights in 2nd layer of MobileNetV2 (ImageNet)

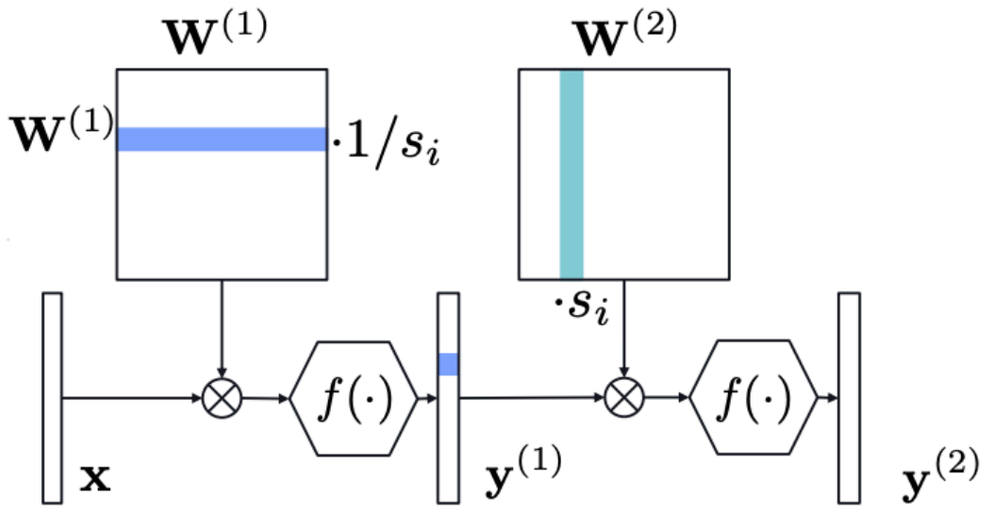
Cross-layer equalization scales weights in neighboring layers for better quantization



$$\mathrm{ReLU}(x) = \max(0, x)$$

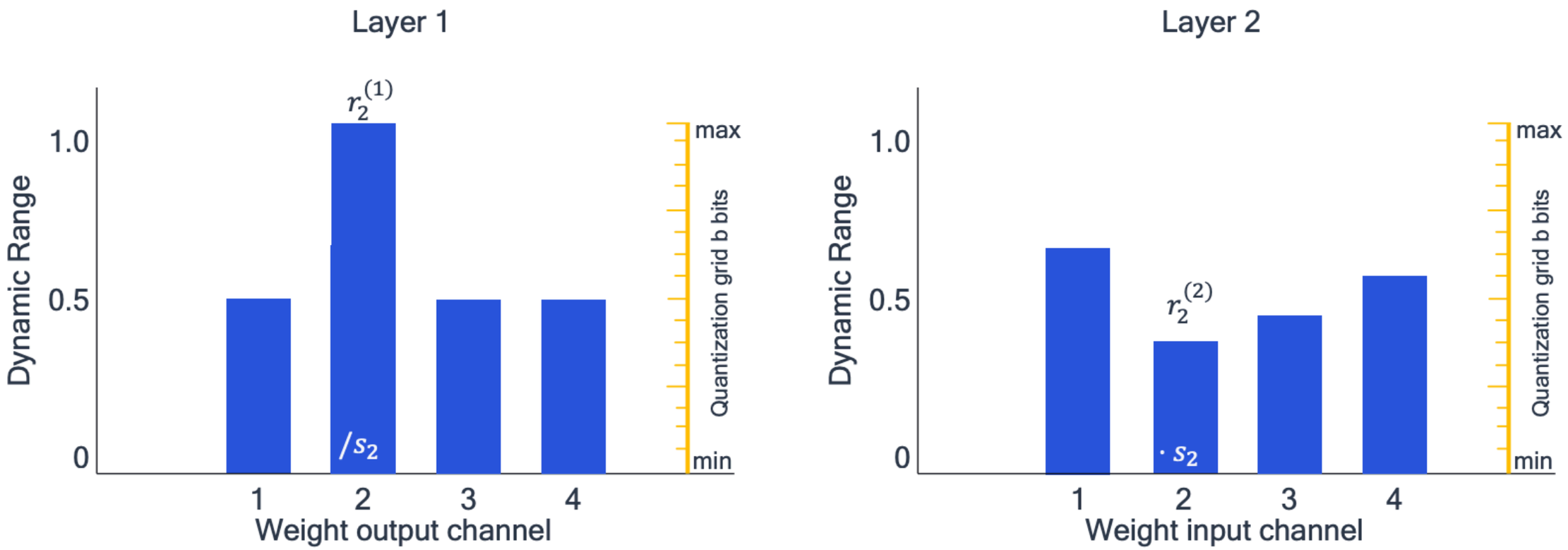
ReLU is scale-equivariant

$$\mathrm{ReLU}(\boldsymbol{s}x) = \boldsymbol{s} \cdot \mathrm{ReLU}(x)$$



We can scale two neighboring layers together to optimize it for quantization

Finding the scaling factors for cross-layer equalization



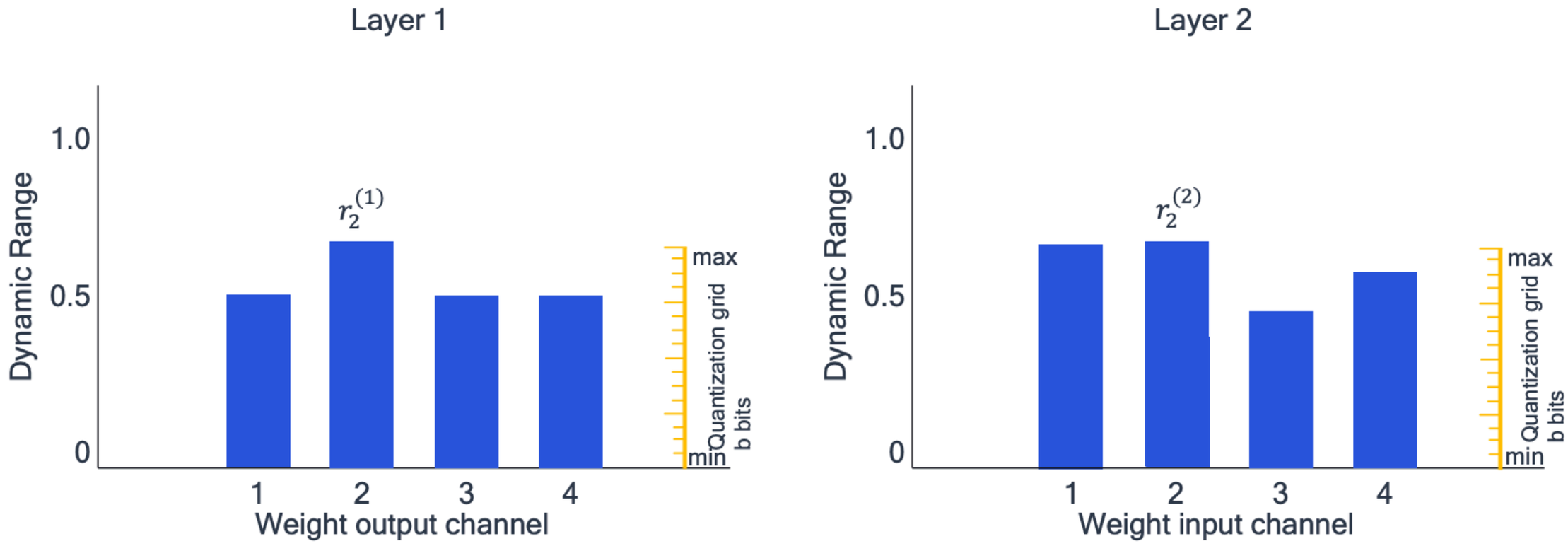
Equalize the weight channels of layer 1 with weight channel of layer 2

by setting $s_i = \frac{1}{r_i^{(2)}} \sqrt{r_i^{(1)} r_i^{(2)}}$

47

Proved to be optimal r_i :Range of layer i

Finding the scaling factors for cross-layer equalization

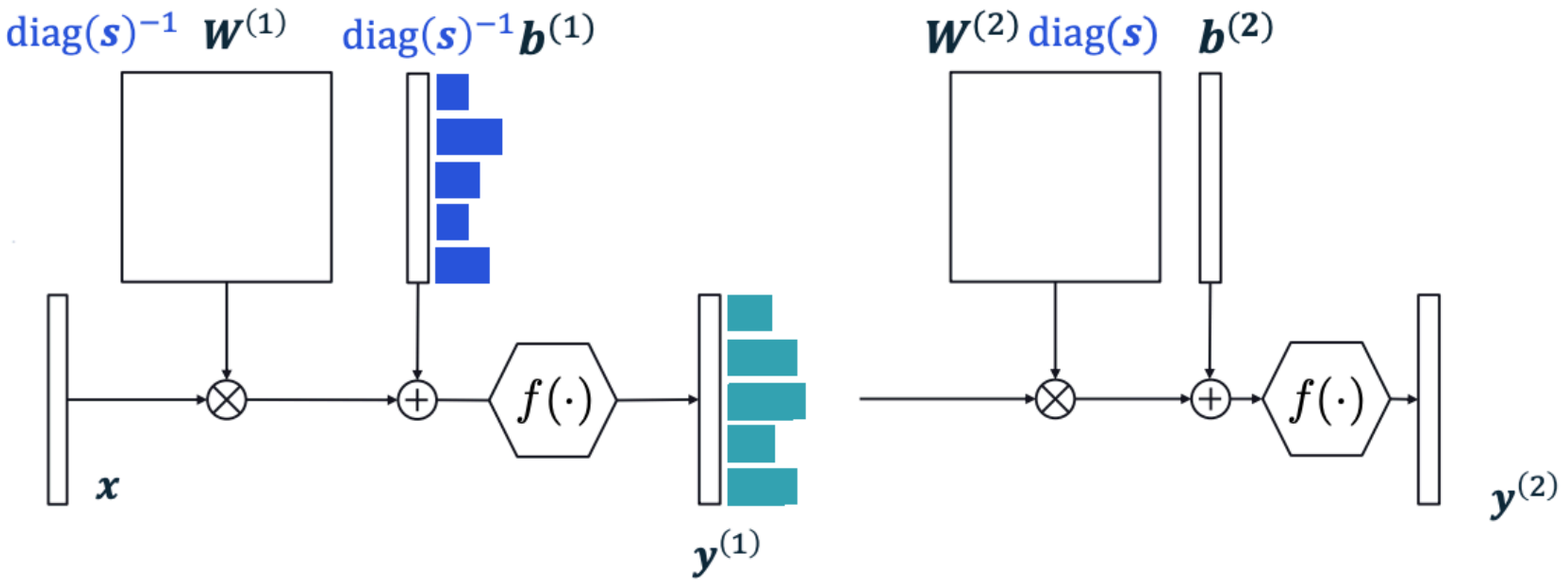


Equalize the weight channels of layer 1 with weight channel of layer 2

by setting $s_i = \frac{1}{r_i^{(2)}} \sqrt{r_i^{(1)} r_i^{(2)}}$

48

Absorbing large biases to the next layer equalizes activation ranges

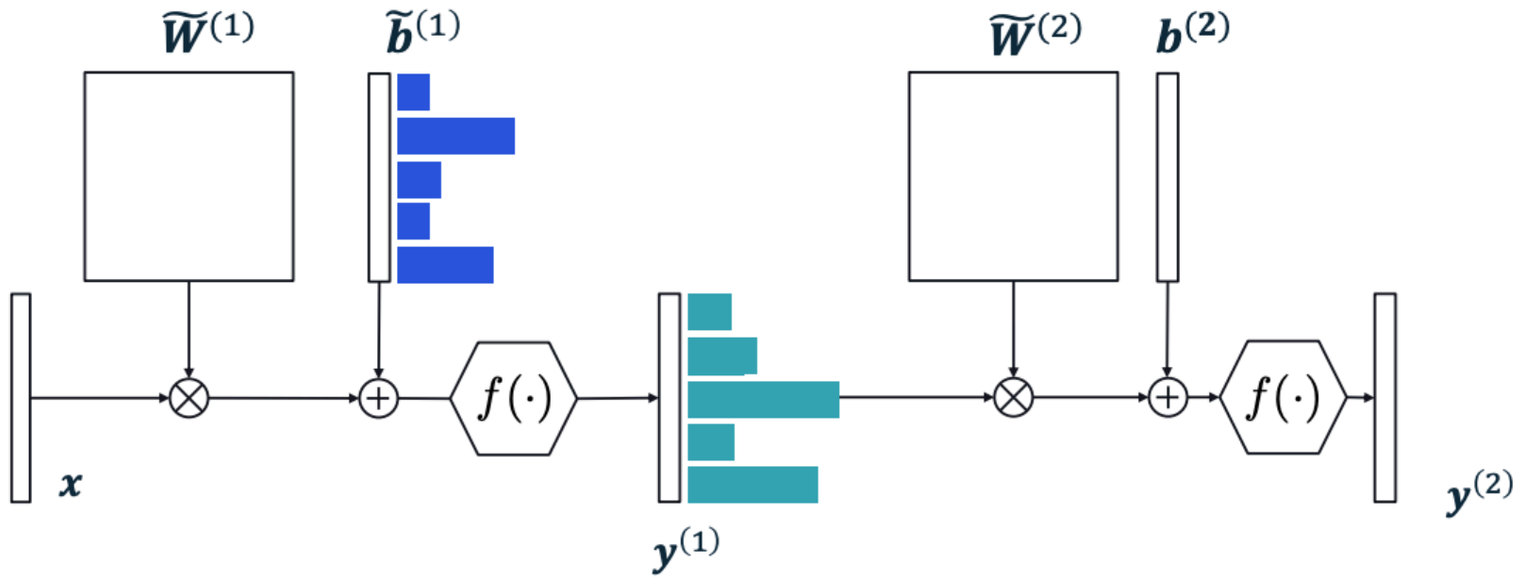


Source sample text

Equaliaze activation ranges by absorbing c from layer 1 into layer 2

49

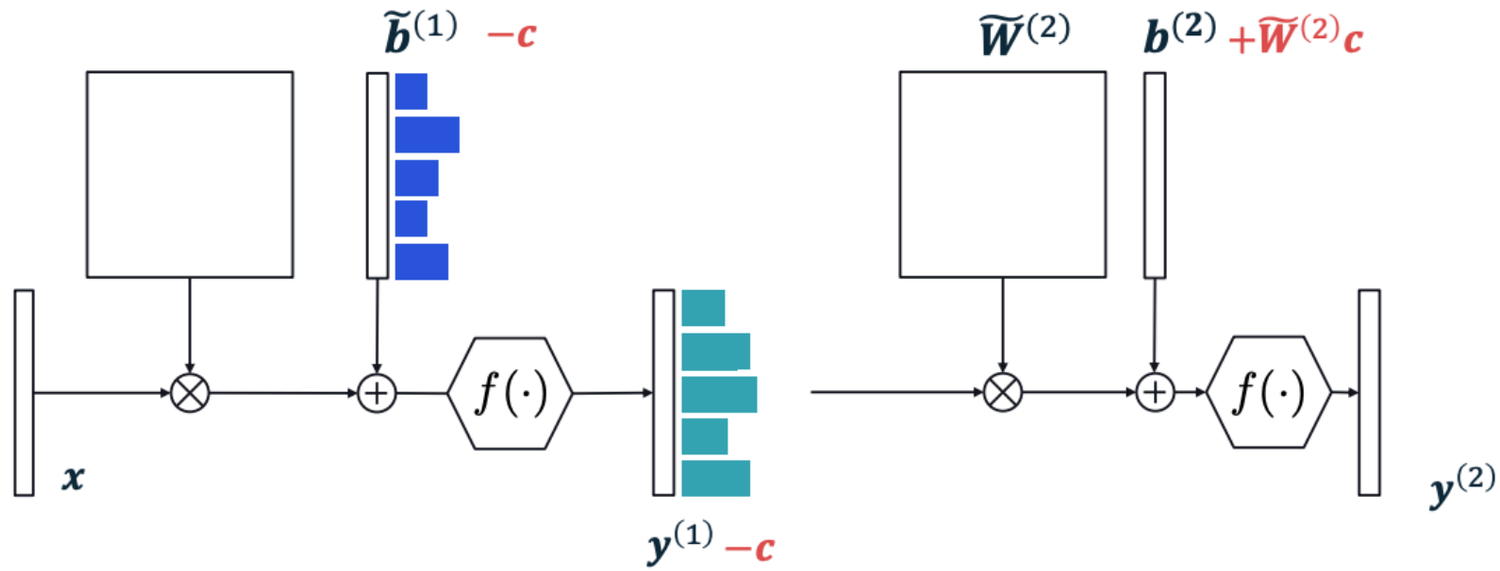
Absorbing large biases to the next layer equalizes activation ranges



When $s_i < 1$, range of biases may increase

Equaliaze activation ranges by absorbing c from layer 1 into layer 2

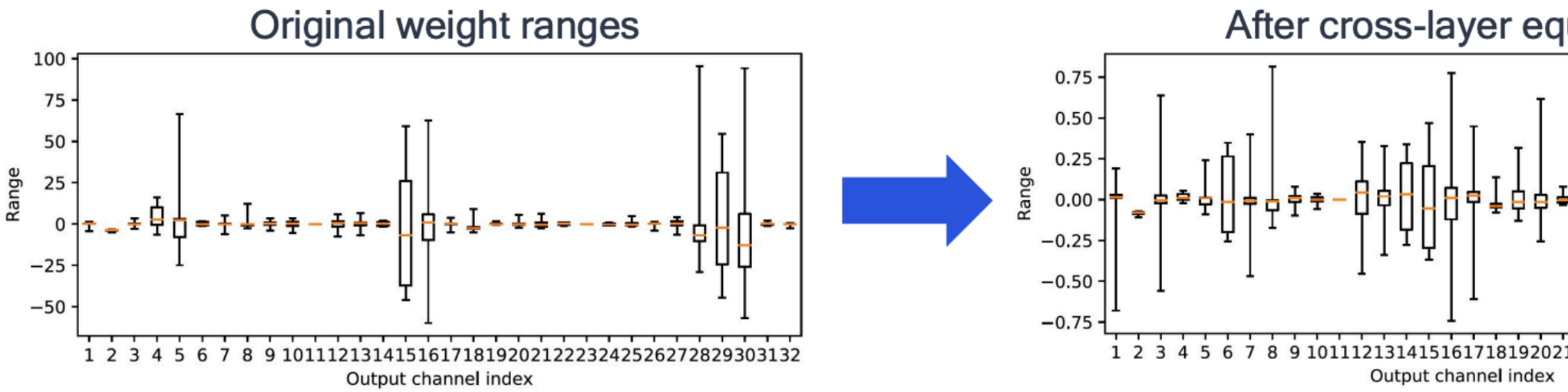
Absorbing large biases to the next layer equalizes activation ranges



When $s_i < 1$, range of biases may increase

Equaliaze activation ranges by absorbing c from layer 1 into layer 2

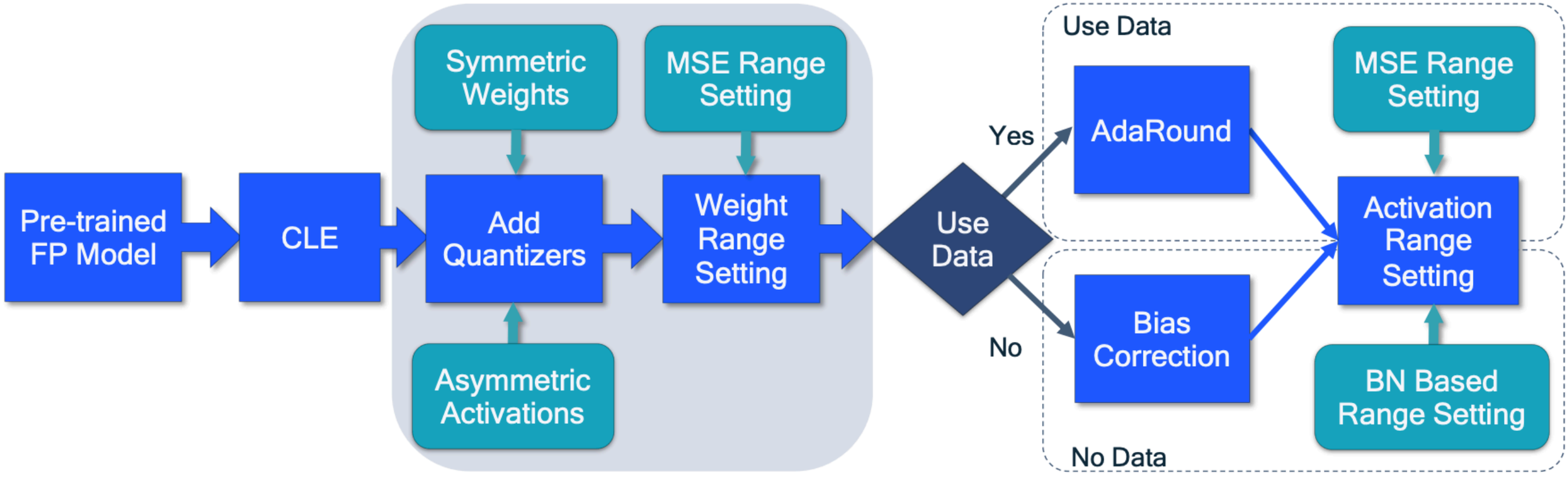
Cross-layer equalization significantly improves accuracy



Model	FP32	INT8
Original Model	71.72	0.12
CLE	71.70	69.91
CLE + absorbing bias	71.57	70.92
Per-channel	71.72	70.65

ImageNet validation accuracy (%) for MobileNetV2

Quantizer and range setting

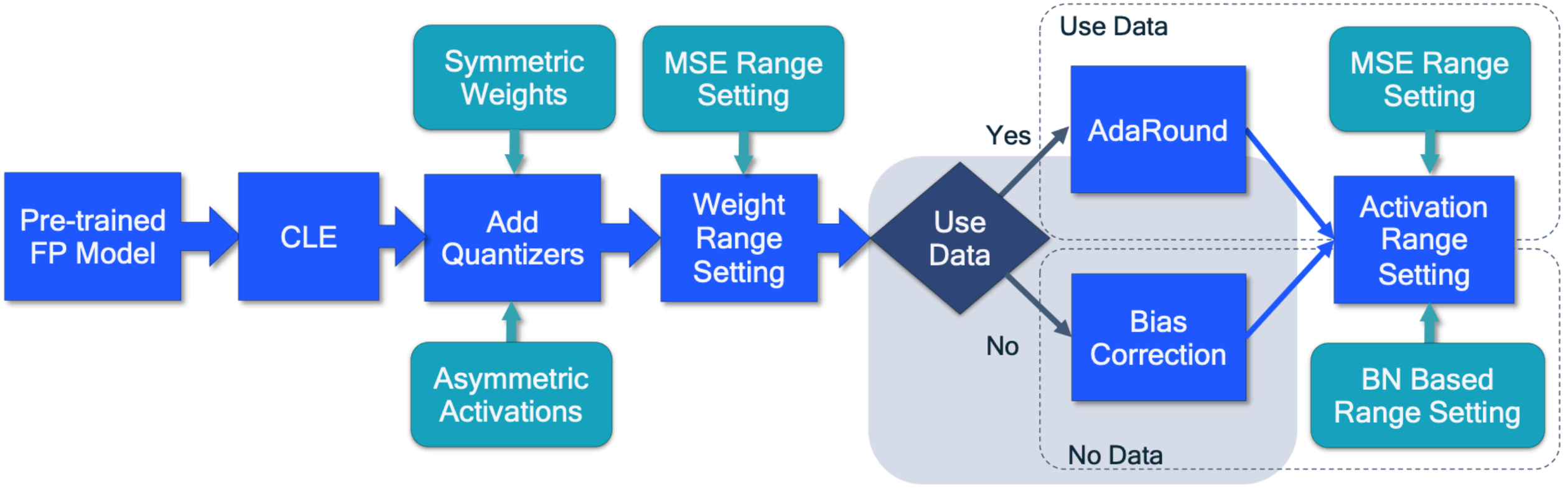


Quantizer and range setting

Model (FP32 Accuracy)	ResNet18 (69.68)		MobileNetV2 (71.72)	
Bit-width	W8	W6	W8	W6
Min-Max	67.57	63.90	71.16	64.48
MSE	69.45	64.64	71.15	65.43
Min-Max (per-channel)	69.60	69.08	71.21	68.52
MSE (per-channel)	69.66	69.24	71.46	68.89

ImageNet validation accuracy (%)

Bias Correction

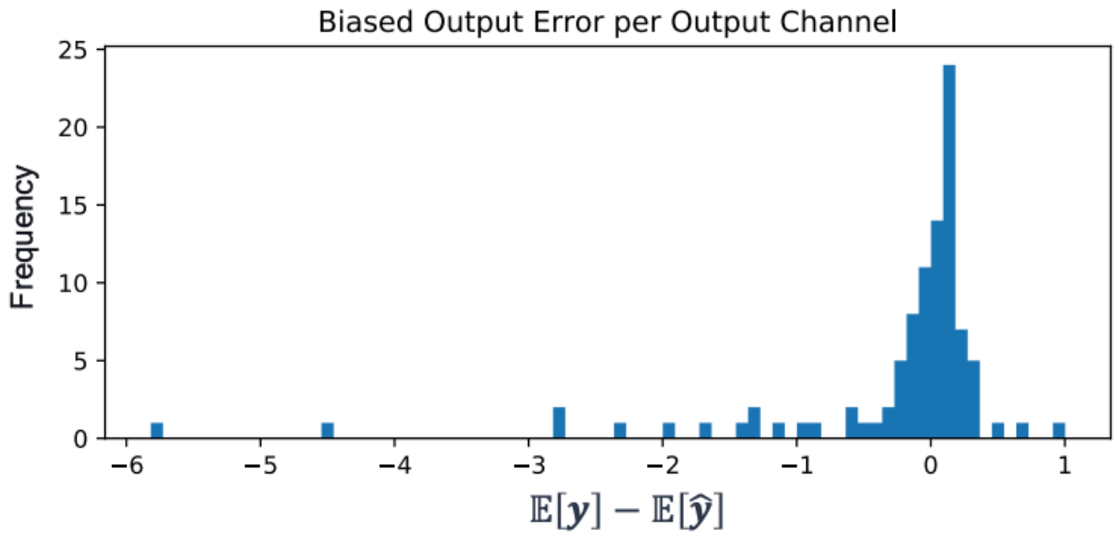


Biased quantization error leads to accuracy drop

Quantization error is often biased

$$\begin{aligned}\mathbb{E}[\boldsymbol{y}] - \mathbb{E}[\widehat{\boldsymbol{y}}] &= \mathbb{E}[\boldsymbol{W}\boldsymbol{x}] - \mathbb{E}[\widehat{\boldsymbol{W}}\boldsymbol{x}] \\ &= \boldsymbol{W}\mathbb{E}[\boldsymbol{x}] - \widehat{\boldsymbol{W}}\mathbb{E}[\boldsymbol{x}] \\ &= \Delta\boldsymbol{W}\ \mathbb{E}[\boldsymbol{x}]\end{aligned}$$

Empirical bias correction



Per-channel biased output error introduced by weight quantization of the second depth-wise separable layer in MobileNetV2

Key idea: Bias correction

data-free

Use batch-norm params
+
Gaussian pre-activations

$$\begin{aligned}\mathbb{E}[\mathbf{x}] &= \mathbb{E}[\mathrm{ReLU}(\mathbf{x}^{\mathrm{pre}})] \\ &= \boldsymbol{\gamma}\mathcal{N}\left(\frac{-\boldsymbol{\beta}}{\boldsymbol{\gamma}}\right) + \boldsymbol{\beta}\left[1 - \Phi\left(\frac{-\boldsymbol{\beta}}{\boldsymbol{\gamma}}\right)\right]\end{aligned}$$

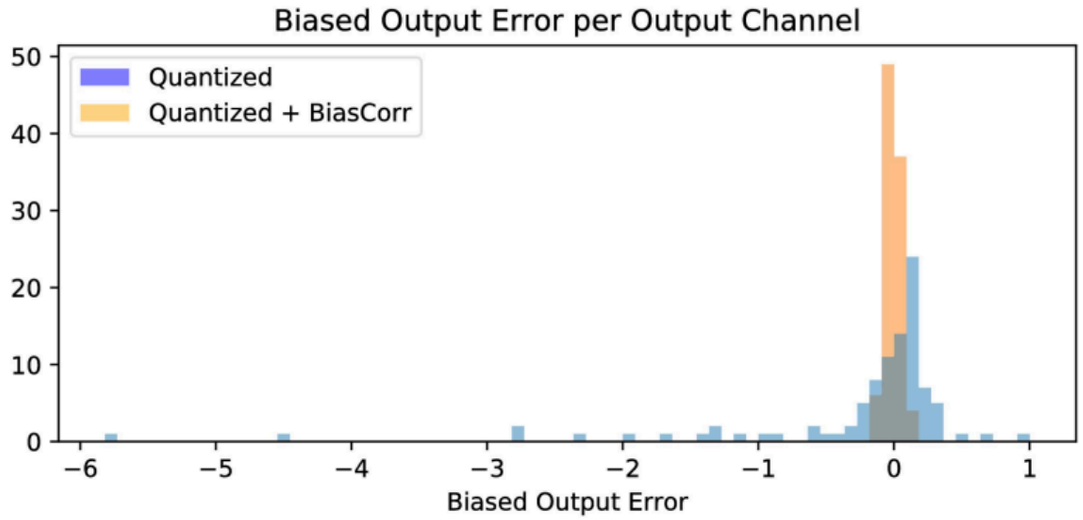
Analytic bias correction

ϕ:normal cdf
β, γ:batch norm parameters

Bias correction

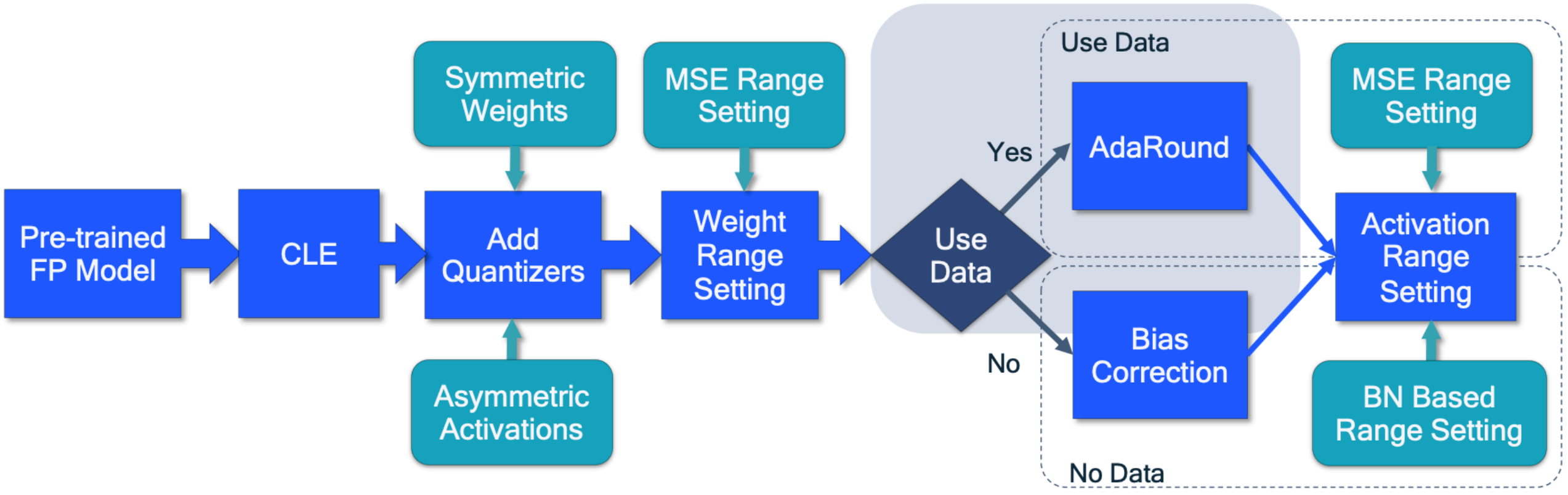
Model	W8A8	FP32
Original Model	0.12	71.72
+bias correction	52.02	71.72
CLE + bias absorption	70.92	71.57
+bias correction	71.79	71.57

ImageNet val. accuracy for MobileNetV2



MobileNetV2 2nd layer

AdaRound


Symmetric Weights
MSE Range Setting
Use Data
MSE Range Setting
Pre-trained FP Model
CLE
Add Quantizers
Weight Range Setting
Use Data
Yes
AdaRound
Activation Range Setting
Asymmetric Activations
No
Bias Correction
BN Based Range Setting
No Data

AdaRound

- Traditionally, in PTQ we use **rounding-to-nearest** operator

$$\boldsymbol{X}_{\text{int}} = \text{clip}\left(\textbf{round}\left(\frac{\boldsymbol{X}}{s}\right) + z, \text{min} = 0, \text{max} = 2^b - 1\right)$$

- However, rounding-to-nearest is not optimal?

Rounding Method	Accuracy (%)
Nearest	52.29
Floor / Ceil	00.10
Stochastic	$52.06^{\pm 5.52}$
Stochastic (best)	**63.06**

4-bit weight quantization of 1st layer of Resnet18, validation accuracy on ImageNet.

Up or Down? Adaptive Rounding for Post-Training Quantization (Nagel, Amjad, et al., ICML 2020)

Up or Down?

How can we systematically find the best rounding choice?

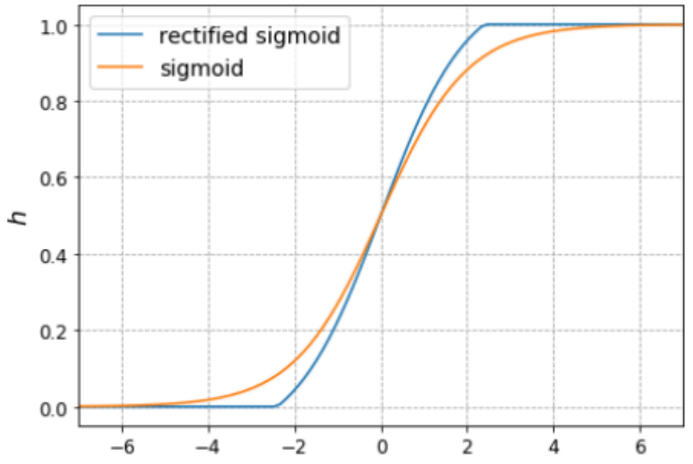
AdaRound: learning to round

AdaRound: learning to round

- Minimize local L_2 loss per-layer rather than task loss:

Surrogate for task loss

$$\arg\min_{\mathbf{V}} \left\| \mathbf{W}\mathbf{x} - \widetilde{\mathbf{W}}\mathbf{x} \right\|_F^2$$



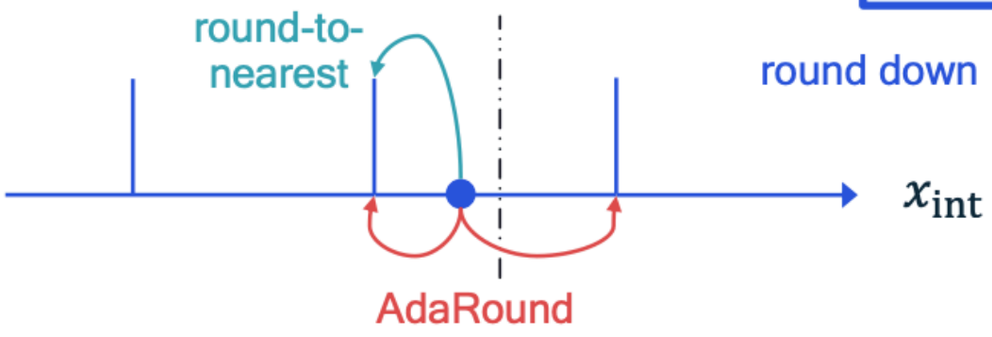
- where $\widetilde{W}$ are soft-quantized weights:

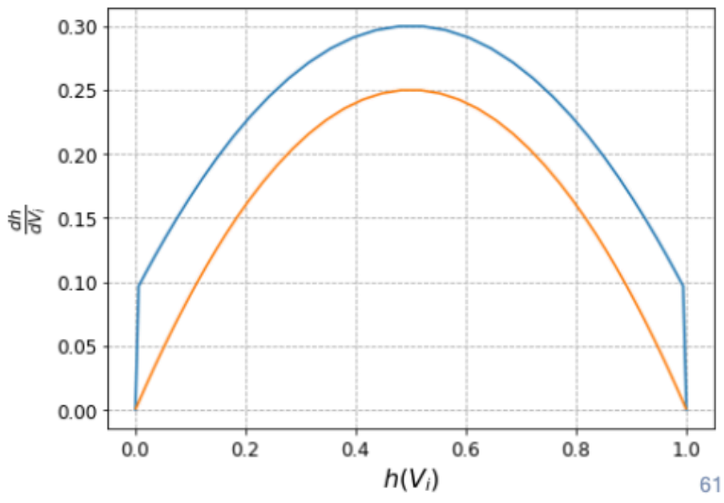
$$\widetilde{\mathbf{W}} = \mathrm{s} \cdot clip\left(\left\lfloor \frac{\mathbf{W}}{\mathrm{s}} \right\rfloor + h(\mathbf{V}), \mathrm{n}, \mathrm{p}\right)$$

$$h(\mathbf{V}) = \mathrm{clip}\left(\sigma(\mathbf{V})(\zeta - \gamma) + \gamma, 0, 1\right)$$

rectified sigmoid

round down + learned value between [0,1]





AdaRound: learning to round

- Minimize local L_2 loss per-layer rather than task loss:

$$\arg\min_{\mathbf{V}} \left\| \mathbf{W}\mathbf{x} - \widetilde{\mathbf{W}}\mathbf{x} \right\|_F^2 + \lambda f_{reg}(\mathbf{V})$$

regularizer forces $h(\mathbf{V})$ to be 0 or 1

- where $\widetilde{W}$ are soft-quantized weights:

$$\widetilde{\mathbf{W}} = \mathbf{s} \cdot clip\left(\left\lfloor \frac{\mathbf{W}}{\mathbf{s}} \right\rfloor + h(\mathbf{V}), \mathbf{n}, \mathbf{p} \right)$$

$$h(\mathbf{V}) = \text{clip}\left(\sigma(\mathbf{V})(\zeta - \gamma) + \gamma, 0, 1\right)$$

rectified sigmoid

round down + learned value between [0,1]

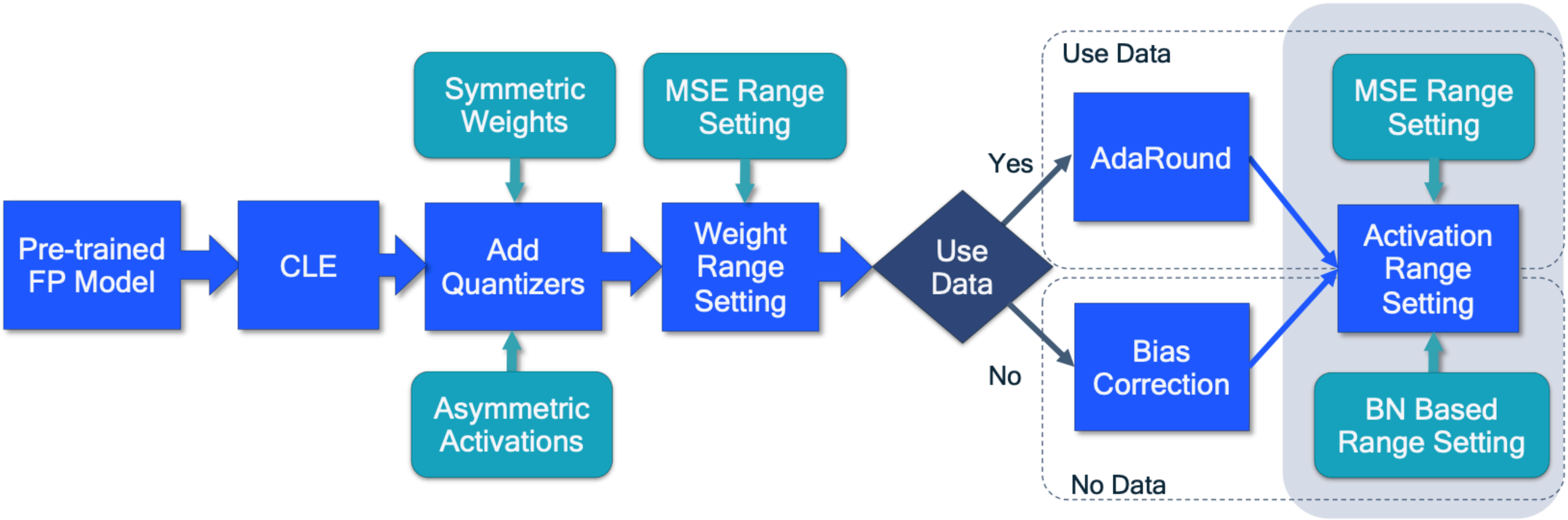
- Regularization:

$$f_{reg}(\mathbf{V}) = \sum_{i,j} 1 - |2h(\mathbf{V}_{i,j}) - 1|^{\beta}$$

AdaRound results

Quantization method	#bits W/A	ResNet18	ResNet50	InceptionV3	MobileNetV2
Full precision	32/32	69.68	76.07	77.40	71.72
CLE + BC	4/8	38.98	52.84	-	46.67
Per channel bias corr*	4*/8	67.4	74.8	59.5	-
AdaRound	4/8	**68.55**	**75.01**	**75.72**	**69.25**

Activation range setting



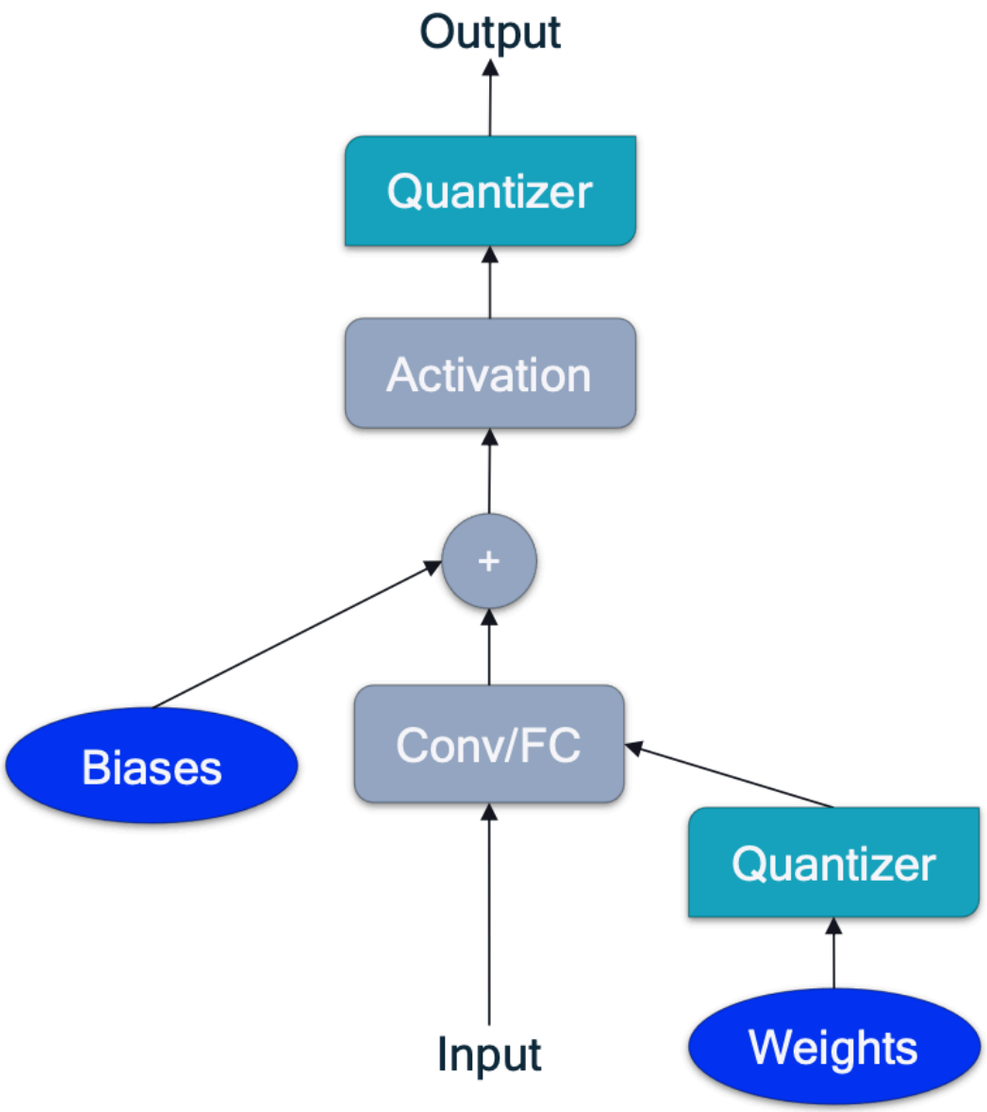
PTQ results using previous pipeline

drop ≤ 1.0%
1.0% < drop ≤1.5%
drop >1.5%

Models	FP32	Per-tensor				Per-channel			
		W8A8	diff	W4A8	diff	W8A8	diff	W4A8	diff
ResNet18	69.68	69.60	-0.08	68.62	-1.06	69.56	-0.12	68.91	-0.77
ResNet50	76.07	75.87	-0.20	75.15	-0.92	75.88	-0.19	75.43	-0.64
MobileNetV2	71.72	70.99	-0.73	69.21	-2.51	71.16	-0.56	69.79	-1.93
InceptionV3	77.40	77.68	+0.28	76.48	-0.92	77.71	-0.31	76.82	-0.58
EfficientNet lite	75.42	75.25	-0.17	71.24	-4.18	75.39	-0.03	74.01	-1.41
DeepLabV3	72.94	72.44	-0.50	70.80	-2.14	72.27	-0.67	71.67	-1.27
EfficientDet-D1	40.08	38.29	-1.79	0.31	-39.77	38.67	-1.41	35.08	-5.00
BERT-base	83.06	82.43	-0.63	81.76	-1.30	82.77	-0.29	82.02	-1.04

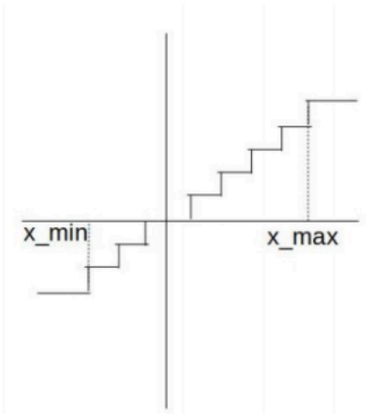
Quantization aware training

Simulating quantization for backward path

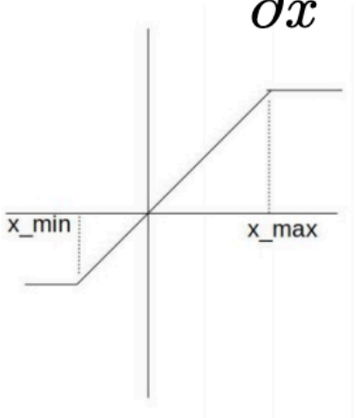


- The round-to-nearest operation does not have meaningful gradients
- Gradient-based training impossible
- Solution: Redefine gradient with the "straight-through estimator" (STE)*

$$\frac{\partial \lfloor x \rceil}{\partial x} = 1$$

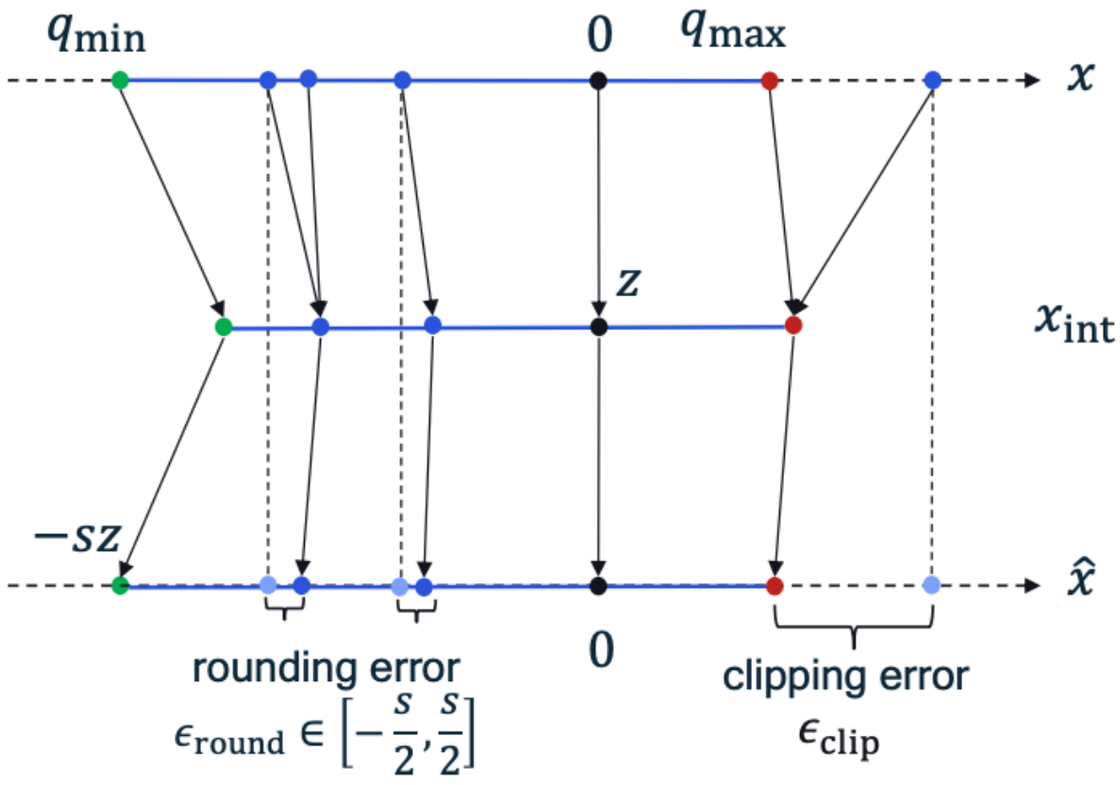


Real Forward pass



Simulated forward pass

Learning the quantization parameters



Learn quantization parameters during training using STE

$$\boldsymbol{X}_{\text{int}} = \text{clamp}\left(\text{round}\left(\frac{\boldsymbol{X}}{s}\right) + z, \min = 0, \max = 2^b - 1\right)$$

$$\hat{\boldsymbol{X}} = s\,(\boldsymbol{X}_{\text{int}} - z)$$

Through task loss gradients, we find the optimal trade-off between ϵ_{clip} & ϵ_{round}

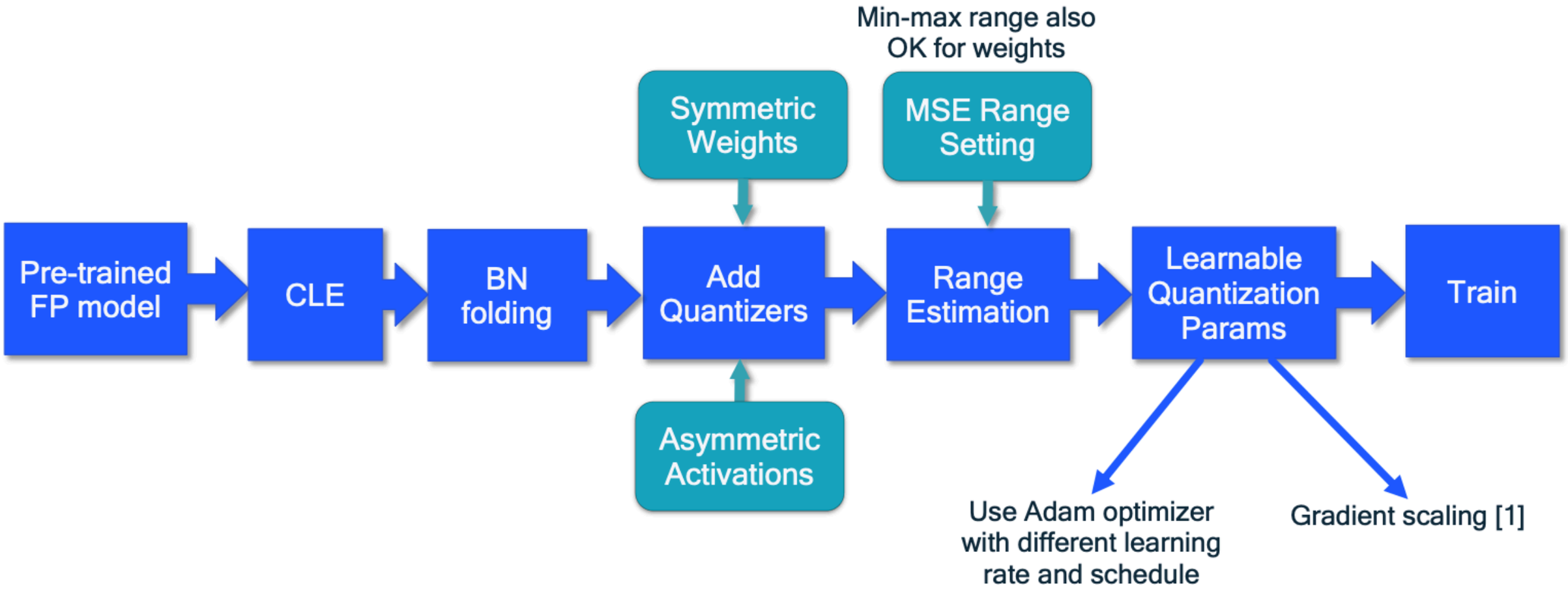
[1] Esser, S. K., McKinstry, J. L., Bablani, D., Appuswamy, R., and Modha, D. S. Learned step size quantization, 2020
[2] Jain, S. R., Gural, A., Wu, M., and Dick, C. Trained uniform quantization for accurate and efficient neural network inference on fixed-point hardware.
[3] Bhalgat, Y., Lee, J., Nagel, M., Blankevoort, T., and Kwak, N. Lsq+: Improving low-bit quantization through learnable offsets and better initialization.

Good initialization matters for QAT

Quantization setting	FP32	PTQ	QAT
W4A8 baseline	71.72	0.10	0.10
W4A8 w/ CLE	71.57	12.99	70.13
W4A8 w/ CLE + BC	71.57	46.90	70.07

Val. accuracy for MobileNetV2 for pet-tensor quantization

Proposed QAT pipeline



[1] Esser, S. K., McKinstry, J. L., Bablani, D., Appuswamy, R., and Modha, D. S. Learned step size quantization, 2020

QAT results

drop ≤ 1.0%
1.0% < drop ≤1.5%
drop >1.5%

Models	FP32	Per-tensor				Per-channel			
		W8A8	diff	W4A8	diff	W8A8	diff	W4A8	diff
ResNet18	69.68	70.38	+0.70	69.76	+0.08	70.43	+0.75	70.01	+0.33
ResNet50	76.07	76.21	+0.14	75.89	-0.18	76.58	+0.51	76.52	+0.45
MobileNetV2	71.72	71.76	+0.04	70.17	-1.55	71.82	+0.10	70.48	-1.24
InceptionV3	77.40	78.33	+0.93	77.84	+0.44	78.45	+1.05	78.12	+0.72
EfficientNet lite	75.42	75.17	-0.25	71.55	-3.87	74.75	-0.67	73.92	-1.50
DeepLabV3	72.94	73.99	+1.05	70.90	-2.04	72.87	-0.07	73.01	+0.07
EfficientDet-D1	40.08	38.94	-1.14	35.34	-4.74	38.97	-1.11	36.75	-3.33
BERT-base	83.06	83.26	+0.20	82.64	-0.42	82.44	-0.62	82.39	-0.67

QAT and PTQ comparison

drop ≤ 1.0%
1.0% < drop ≤1.5%
drop >1.5%

Difference from FP accuracy for W4A8 quantization

Models	FP32	Per-tensor		Per-channel	
		PTQ	QAT	PTQ	QAT
ResNet18	69.68	-1.06	+0.08	-0.77	+0.33
ResNet50	76.07	-0.92	-0.18	-0.64	+0.45
MobileNetV2	71.72	-2.51	-1.55	-1.93	-1.24
InceptionV3	77.40	-0.92	+0.44	-0.58	+0.72
EfficientNet lite	75.42	-4.18	-3.87	-1.41	-1.50
DeepLabV3	72.94	-2.14	-2.04	-1.27	+0.07
EfficientDet-D1	40.08	-39.77	-4.74	-5.00	-3.33
BERT-base	83.06	-1.30	-0.42	-1.04	-0.67

Relaxed Quantization for Discretized Neural Networks (Louizos, et al.)	ICLR 2019
Data-Free Quantization Through Weight Equalization and Bias Correction (Nagel, van Baalen, et al.)	ICCV 2019
Up or Down? Adaptive Rounding for Post-Training Quantization (Nagel, Amjad, et al.)	ICML 2020
Bayesian Bits: Unifying Quantization and Pruning (van Baalen, Louizos, et al.)	NeurIPS 2021
In-Hindsight Quantization Range Estimation for Quantized Training (Fournarakis, et al.)	CVPR 2021
A White Paper on Neural Network Quantization (Nagel, Fournarakis, et al.)	ArXiv 2021
Understanding and Overcoming the Challenges of Efficient Transformer Quantization (Bondarenko, et al.)	EMNLP 2021

Source sample text

Leading research in quantization