

Model Compression Quantization

Instructor: Siting Liu

School of Information Science and Technology (SIST), ShanghaiTech University

Shanghai Institute of Semiconductor Innovation (SISI)

<https://faculty.sist.shanghaitech.edu.cn/liust/EE219/EE219.html>



Computer Architecture - EE219
faculty.sist.shanghaitech.edu.cn

立志成才 报国裕民



Review

- Basics on machine learning
- Various neural network architectures
 - Fully-connected layers (matrix multiplication)
 - Convolution layers (2D/3D convolution)
 - Activation functions (ReLU & special functions)
 - Normalization (Batch/layer/RMS-norm, special functions, etc.)
 - Matrix reshape (transpose, vectorize)
- Next step: How to compute?



Model Compression: Motivation

```
sitingliu — python — 64x20
(base) sitingliu@Sittings-MacBook-Air ~ % python
Python 3.9.13 (main, Aug 25 2022, 18:29:29)
[Clang 12.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 0.1+0.2
```



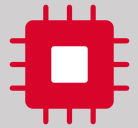
Model Compression: Motivation

```
sitingliu — python — 64x20
(base) sitingliu@Sittings-MacBook-Air ~ % python
Python 3.9.13 (main, Aug 25 2022, 18:29:29)
[Clang 12.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 0.1+0.2
0.30000000000000004
>>> 
```



Model Compression: Motivation (Cont'd)

The End of "Free Lunch": Challenges in the AI Era



Computational Explosion



The Memory Wall



The Power Wall

In AI Era, many applications feature inherent error resilience, such as vision, speech, language tasks

"Good Enough" Computing — Instead of always performing exact, energy-intensive computations, we intentionally introduce controlled, bounded approximations when the application can tolerate them.



Accuracy

Trade off



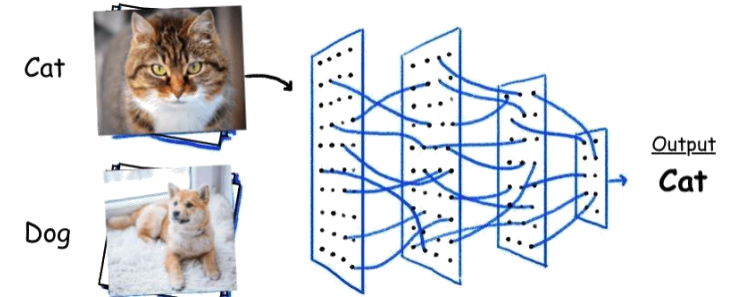
Energy Efficiency



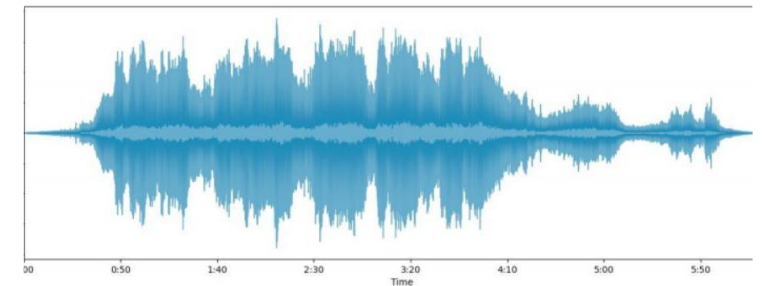
Performance



Image Recognition



Speech Recognition



Natural Language Processing



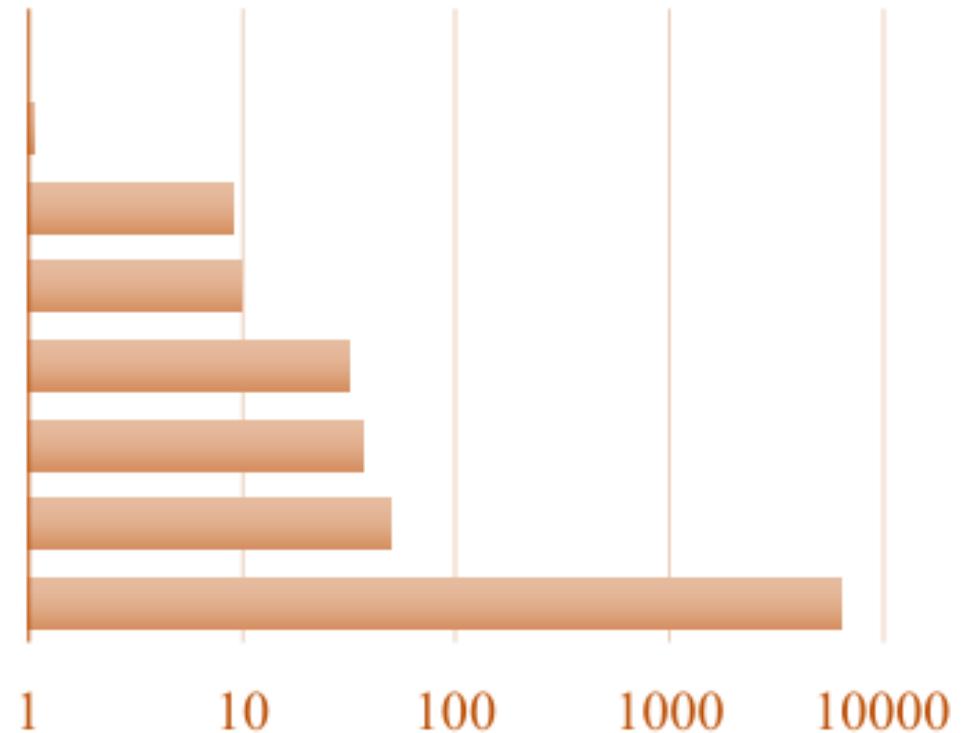
民生

Model Compression: Quantization

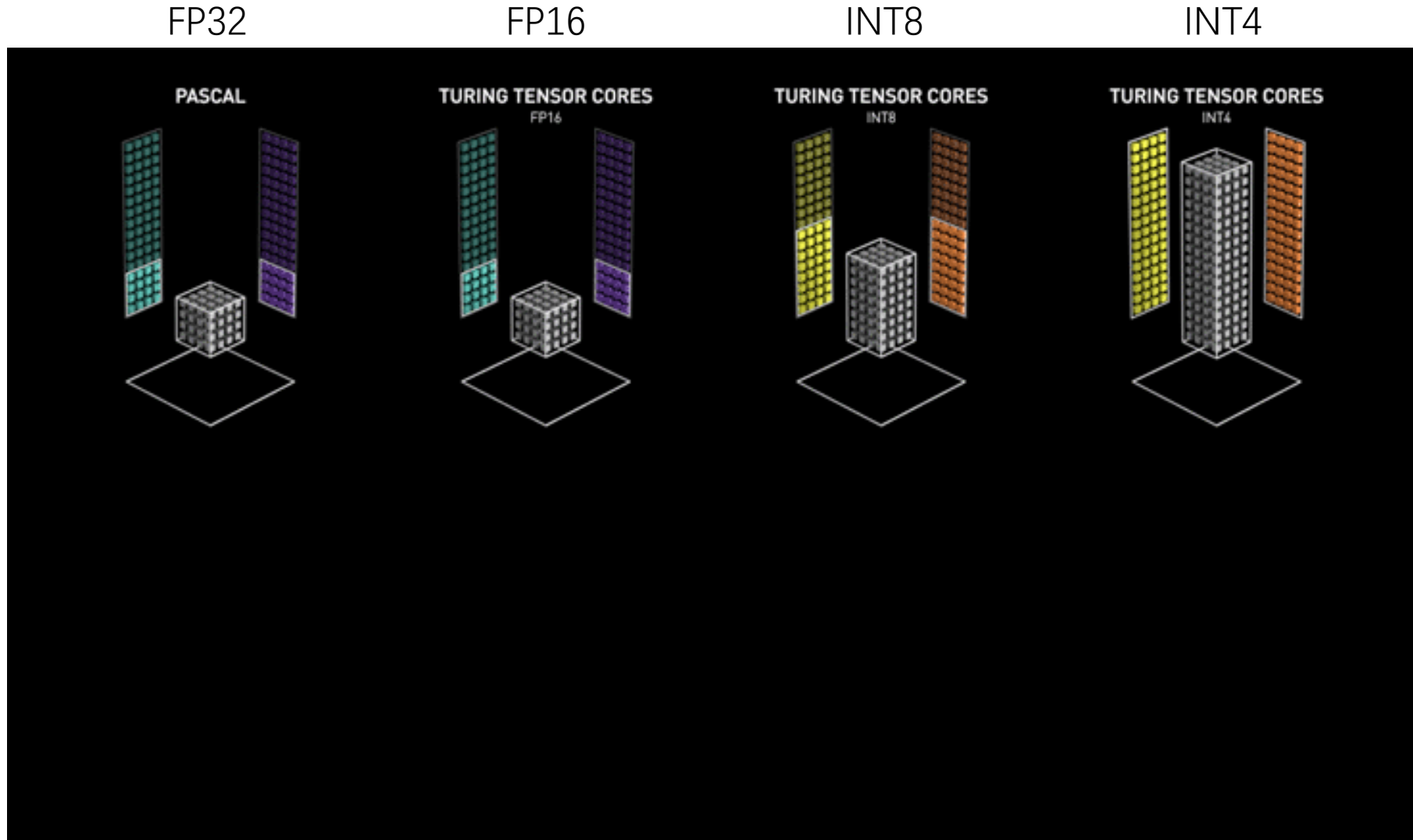
- Reduce model parameters & computations

Operation	Energy (pJ)
32b int ADD	0.1
32b float ADD	0.9
32b Reg. file	1
32b int MULTI	31
32b float MULTI	37
32b SRAM	5
32b DRAM	640

Relative Energy Cost



Model Compression: Quantization

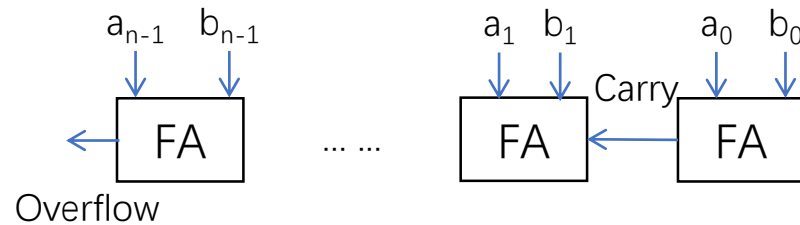


Credit to Nvidia

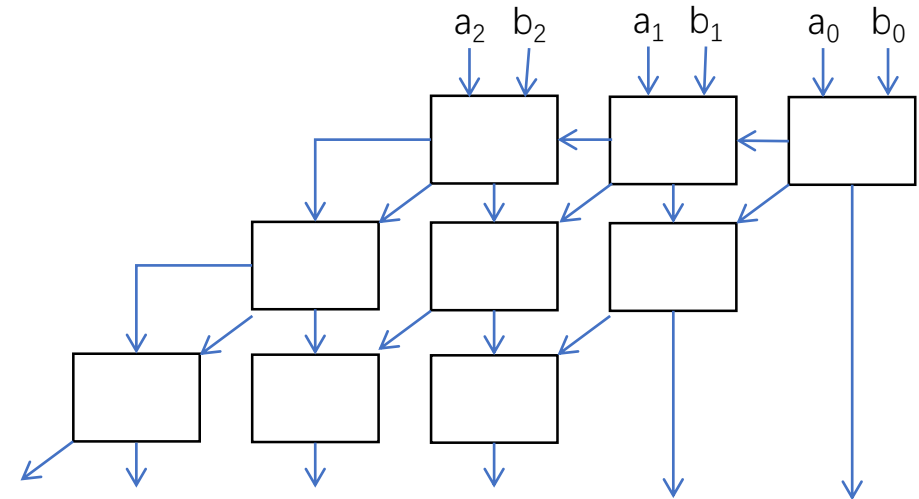
立志成才 报国裕民

Quantization: Number Representation Review

- Integer
 - Unsigned
 - Signed (2's complement)
 - Simple arithmetic



n-bit RCA
 $O(n)$

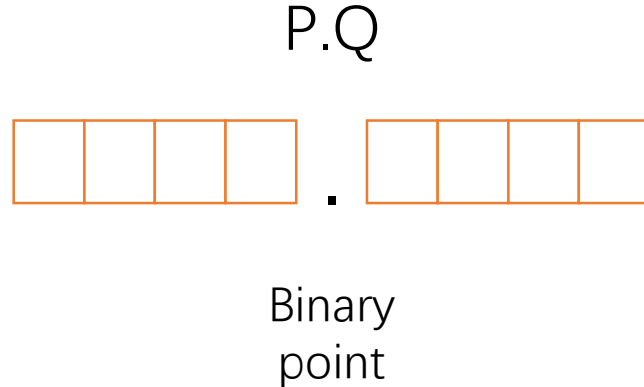


Array multiplier
 $O(n^2)$



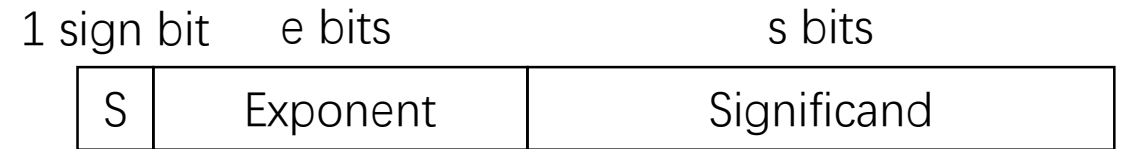
Fractions

- Fixed-point
 - Scaled integers



Need to record the position
of the binary point,
especially for multiplications

- Floating-point
 - IEEE 754 standard



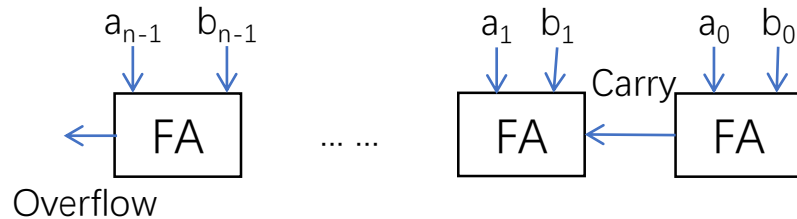
$$(-1)^S \times (1. \text{Significand})_2 \times 2^{\text{Exponent} - \text{bias}}$$

Half-precision	e=5	s=10	bias=15
Single-precision	e=8	s=23	bias=127
Double-precision	e=11	s=52	bias=1023

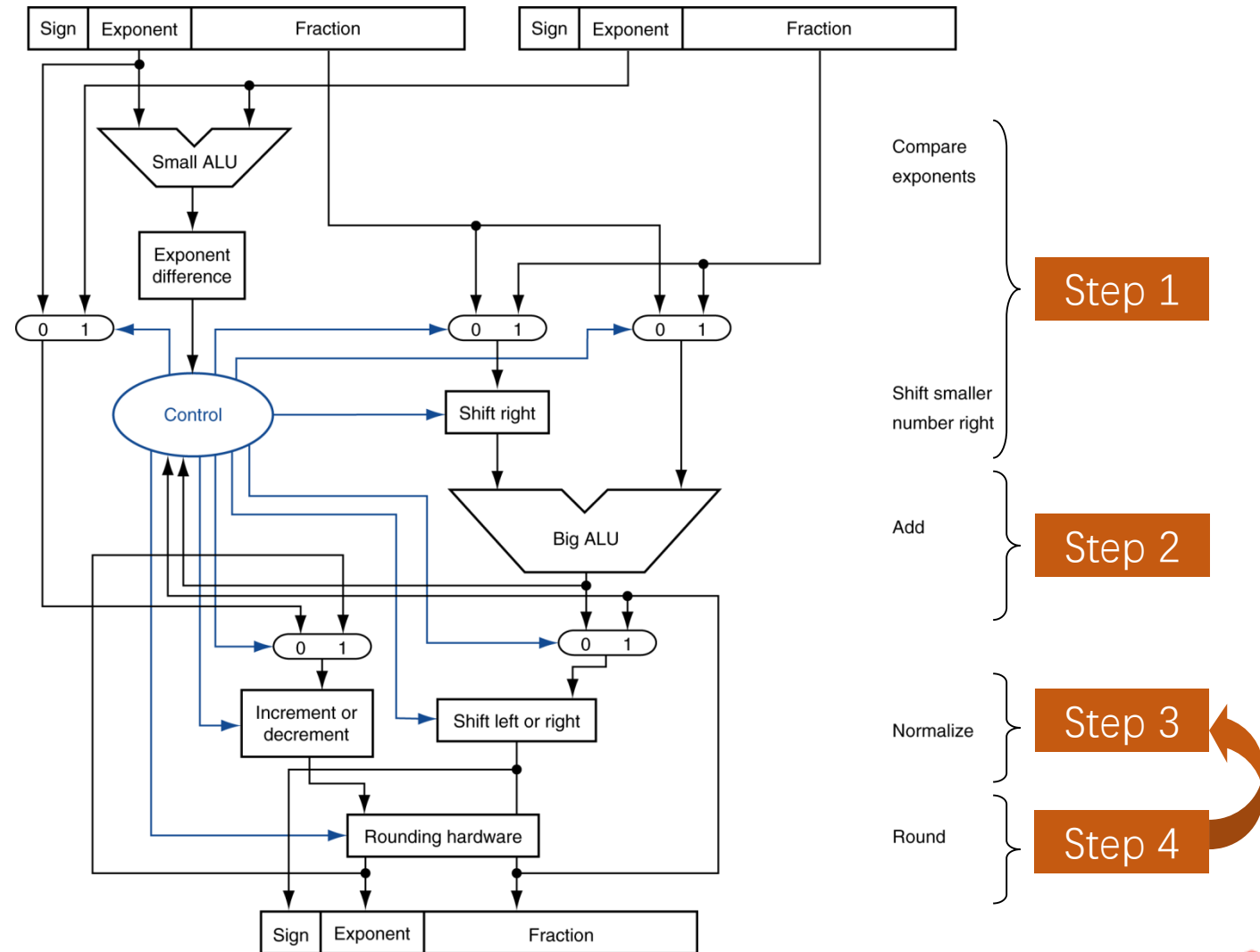
- Large dynamic range with limited bits
- Scale-invariant relative precision
- Auto-scaling for different ranges
- Complex datapath for arithmetic

Floating-point Arithmetic

- Floating-point addition
 1. Alignment shift/pre-shift
 2. Add/subtract the significands
 3. Normalization shift/post-shift
 4. Round
 5. Repeat 3,4 if not normalized



n-bit RCA
 $O(n)$



Floating-point Arithmetic (Cont'd)

- Floating-point Multiplications
 1. Add exponent (minus bias)
 2. Multiply significands
 3. Normalize result & check for over/underflow
 4. Round and renormalize if necessary
 5. Determine sign: $s_1 \oplus s_2 \Rightarrow s$
- Different rounding schemes
 - Round to zero
 - Downward
 - Upward
 - To-nearest (ties-to-even)



Hardware Implications

- Rough energy numbers (in 45-nm technology node) from “Computing’s Energy Problem, M. Horowitz, ISSCC, 2014”

INT		
ADD		
8 bit		0.03 pJ
32 bit		0.1 pJ
MULTI		
8 bit		0.2 pJ
32 bit		3 pJ

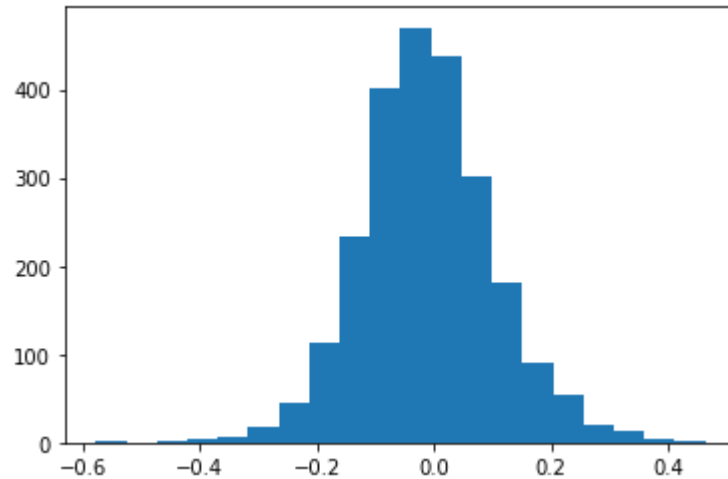
FP		
FADD		
16 bit		0.4 pJ
32 bit		0.9 pJ
FMULTI		
16 bit		1 pJ
32 bit		4 pJ

Memory		
Cache		(64 bit)
8 KB		10 pJ
32 KB		20 pJ
1 MB		100 pJ
DRAM		1.3-2.6 nJ

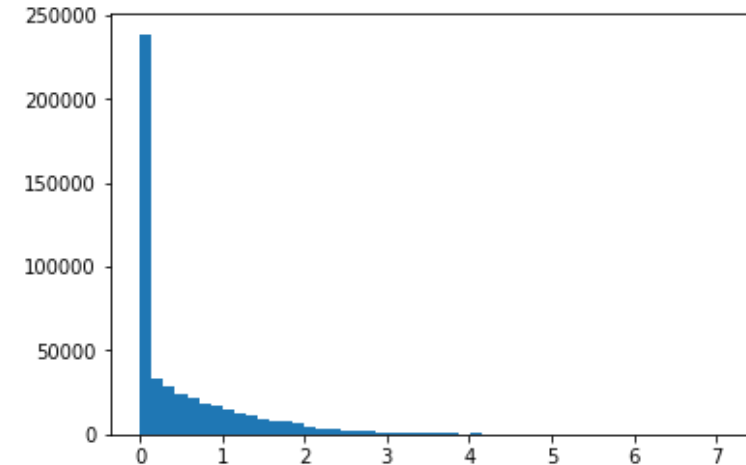


Data Distribution

- Complex floating-point format is not needed with limited data range;
- Trade precision/accuracy for performance and energy savings.



Weight distribution @ CNN layer



Act. distribution @ CNN layer



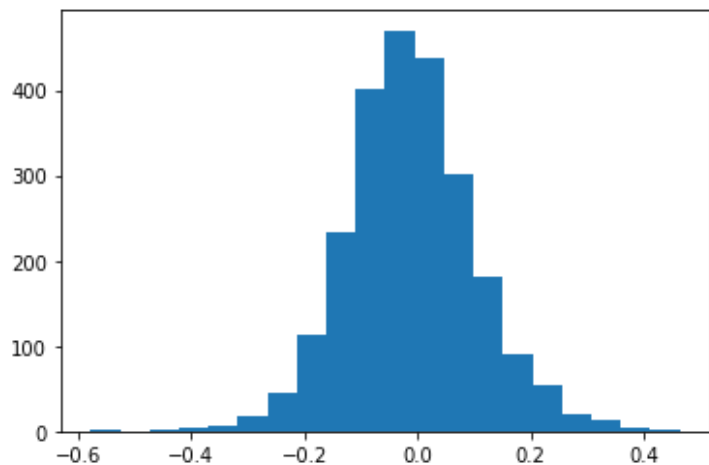
From Floating-point to Integer

- Complex floating-point format is not needed with limited data range;
- Trade precision/accuracy for performance and energy savings.
- Data compressed, may lead to accuracy loss, especially when bit width is low
- Resolution is lower due to the reduced number of available binary patterns

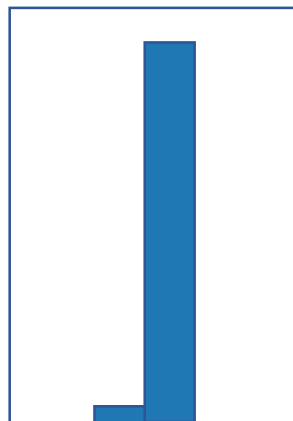
	Dynamic Range	Min positive Value
FP32	$-3.4 \times 10^{38} \sim +3.4 \times 10^{38}$	1.4×10^{-45}
FP16	$-65504 \sim +65504$	5.96×10^{-8}
INT8	$-128 \sim +127$	1



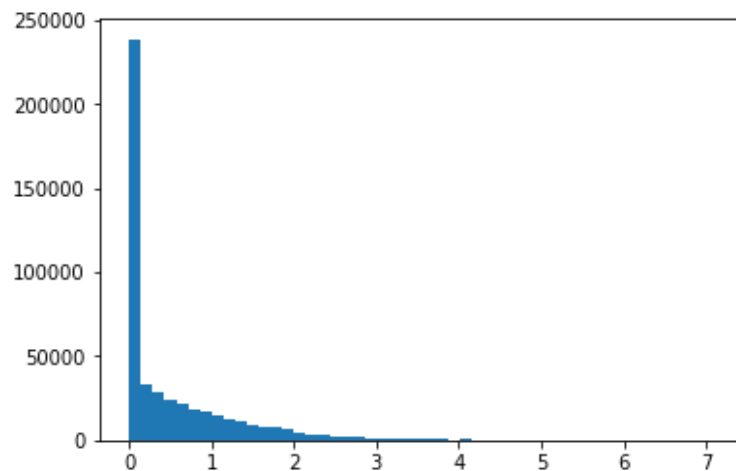
Possible Accuracy Loss



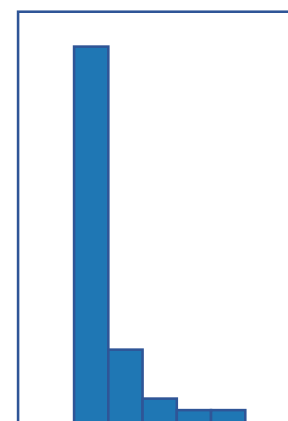
Round to
integer



-1 0



Round to
integer

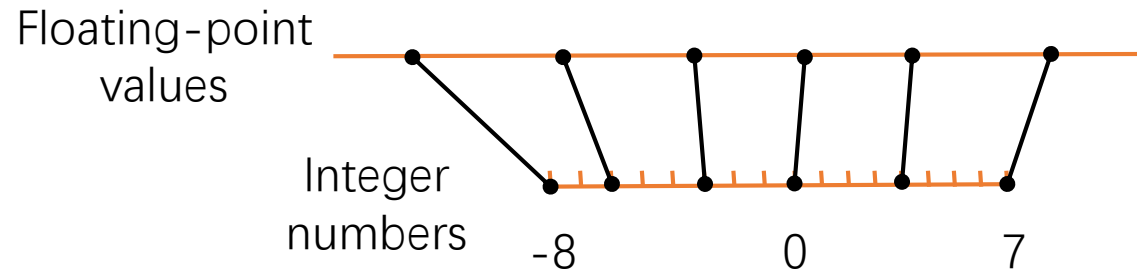


0 1 2 3 4

立志成才 报国裕民

Quantization

- Problem statement: Mapping floating-point values to discrete integer values.



- Need to decide
 - Bit-width (BW) of the integer value
 - Mapping scheme
- Intuitively
 - Larger BW $\rightarrow$ higher precision
 - Use as many integer values as possible



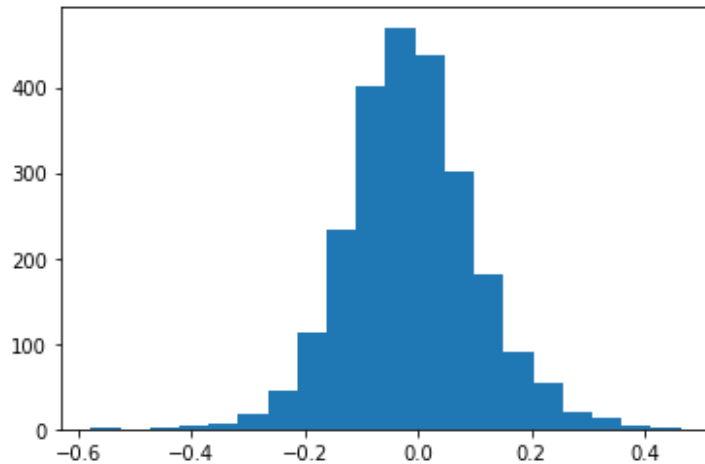
A Simple Mapping Scheme

- Uniform (linear) & symmetric quantization

$$x_{int} = \text{round}\left(\frac{x}{s_x}\right)$$

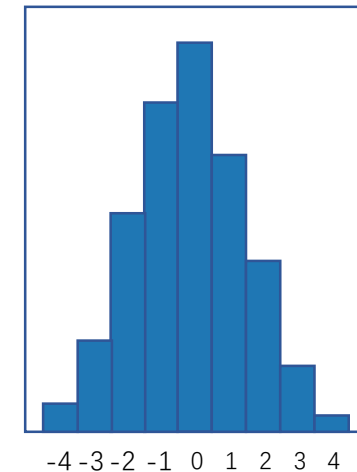
Scaling factor

- Example



***10**

Scale and round



Matrix Multiplication with Quantized Numbers

$$X = \begin{pmatrix} 0.5 & 0.7 \\ 0.4 & 0.5 \end{pmatrix} \quad W = \begin{pmatrix} 0.3 & 0.3 \\ 0.3 & 0.9 \end{pmatrix} \quad \Rightarrow \quad X_{int} = \begin{pmatrix} 5 & 7 \\ 4 & 5 \end{pmatrix} \quad W_{int} = \begin{pmatrix} 3 & 3 \\ 3 & 9 \end{pmatrix}$$

Quantize with
scaling factor 1/10

$$V = XW = \begin{pmatrix} 0.36 & 0.78 \\ 0.27 & 0.57 \end{pmatrix} \quad \Leftarrow \quad V_{int} = X_{int}W_{int} = \begin{pmatrix} 36 & 78 \\ 27 & 57 \end{pmatrix}$$

Dequantize with scaling factor
 $s_X s_W = 1/100$

- What if $X = \begin{pmatrix} 0.51 & 0.7 \\ 0.4 & 0.5 \end{pmatrix}$?

$$X_{int} = \begin{pmatrix} 5 & 7 \\ 4 & 5 \end{pmatrix} \quad W_{int} = \begin{pmatrix} 3 & 3 \\ 3 & 9 \end{pmatrix}$$

$$V = XW = \begin{pmatrix} 0.363 & 0.783 \\ 0.270 & 0.570 \end{pmatrix} \quad \hat{V} = XW = \begin{pmatrix} 0.36 & 0.78 \\ 0.27 & 0.57 \end{pmatrix} \quad V_{int} = X_{int}W_{int} = \begin{pmatrix} 36 & 78 \\ 27 & 57 \end{pmatrix}$$

Can assign $s_X = 1/100$, but require
extra digit to represent the value;
otherwise leads to overflow

Matrix Multiplication (Cont'd)

- What if BW=1 in decimal, $X = \begin{pmatrix} 0.95 & 0.7 \\ 0.4 & 0.5 \end{pmatrix}$ $W = \begin{pmatrix} 0.3 & 0.3 \\ 0.3 & 0.9 \end{pmatrix}$

$$x_{int} = \text{clip} \left(\text{round} \left(\frac{x}{s_x} \right); 0, \text{base}^{BW} - 1 \right)$$

Base=10 in this example;
Base=2 in real computer

Quantize with scaling
factor 1/10 and clip

$$X_{int} = \begin{pmatrix} 9.0 & 7.0 \\ 4.0 & 5.0 \end{pmatrix} \quad W_{int} = \begin{pmatrix} 3 & 3 \\ 3 & 9 \end{pmatrix}$$

$$V_{int} = X_{int} W_{int} = \begin{pmatrix} 48 & 90 \\ 27 & 57 \end{pmatrix}$$

$$V = XW = \begin{pmatrix} 0.495 & 0.915 \\ 0.27 & 0.57 \end{pmatrix}$$

$$\hat{V} = XW = \begin{pmatrix} 0.48 & 0.90 \\ 0.27 & 0.57 \end{pmatrix}$$



Dequantize with scaling factor
 $s_X s_W = 1/100$

- Alternatively, $s_X = \frac{0.95}{9}$

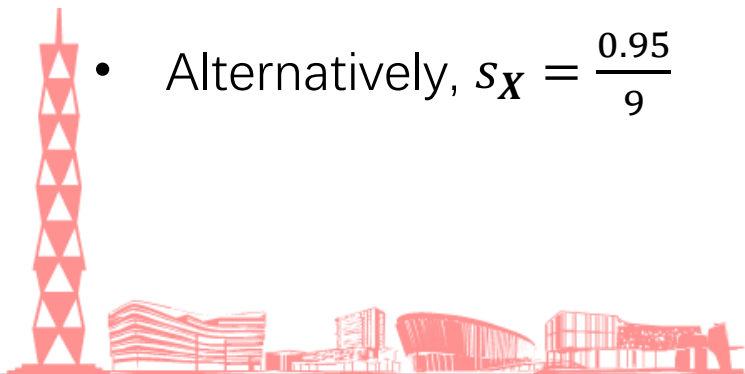
$$X_{int} = \begin{pmatrix} 9.0 & 6.6 \\ 3.8 & 4.7 \end{pmatrix} \quad W_{int} = \begin{pmatrix} 3 & 3 \\ 3 & 9 \end{pmatrix}$$

$$V_{int} = X_{int} W_{int} = \begin{pmatrix} 46.8 & 86.4 \\ 25.5 & 53.7 \end{pmatrix}$$

$$\hat{V} = XW = \begin{pmatrix} 0.494 & 0.912 \\ 0.269 & 0.567 \end{pmatrix}$$



Dequantize with scaling factor
 $s_X s_W = 0.95/90$



立志成才 报国裕民

Schemes for Scaling Factor Optimization

- Use the upper/lower bound to decide the scaling factor (min-max)

- Unsigned

$$s_x = \frac{\text{MAX}}{2^{\text{BW}} - 1}$$

- Signed

$$s_x = \max\left(\frac{|\text{Negative MIN}|}{2^{\text{BW}-1}}, \frac{\text{Positive MAX}}{2^{\text{BW}-1} - 1}\right)$$

- How to decide MIN and MAX?

- During inference, weights: directly obtained from the pre-trained weights;  **No clip error**
- During inference, activations: usually obtained by the MIN and MAX of a calibration dataset;
- Above is usually used in static post-training quantization (PTQ)
- Can be applied to different granularity: layer-wise, channel-wise, group-wise, etc.



Other Schemes for Scaling Factor Optimization

- MSE-based scaling factor optimization
 - Minimize the L2 difference between the true results and dequantized results

$$\arg \min_{s_W \text{ or range}} |W - \hat{W}|^2$$

$$\arg \min_{s_X \text{ or range}} |X - \hat{X}|^2$$

$$\arg \min_{s_X, s_W, \text{etc.}} |XW - \hat{X}\hat{W}|^2$$

Simulated
quantization or
dequantization
 $\hat{x} = x_{int} * s_x$

- BN-based scaling factor selection
 - If a layer has batch-normalized activations, the per-channel mean and standard deviation of the activations are equal to the learned batch normalization shift and scale parameters, respectively.
 - Can be calibration/data-free
- Entropy-based method
 - Minimize the data distribution difference between the true values and dequantized values

Entropy-based Scaling Factor Optimization

- Minimize relative entropy/Kullback-Leibler divergence (KL-div), applied to activations only
 - P, Q - two discrete probability distributions
 - $\text{KL-div} (P , Q) = \text{SUM} (P[i] * \log (P[i] / Q[i]) , i)$
 - For each layer:
 - collect histograms of activations.
 - generate many quantized distributions with different saturation thresholds.
 - pick threshold which minimizes $\text{KL-div}(x, x_q)$



Entropy-based Scaling Factor Optimization (Cont'd)

- Pseudo-code (apply to activations only)

Input: FP32 histogram H with 2048 bins: $\text{bin}[0], \dots, \text{bin}[2047]$

For i in range(128 , 2048):

$\text{reference_distribution_P} = [\text{bin}[0], \dots, \text{bin}[i-1]]$ // take first 'i' bins from H

$\text{outliers_count} = \text{sum}(\text{bin}[i], \text{bin}[i+1], \dots, \text{bin}[2047])$

$\text{reference_distribution_P}[i-1] += \text{outliers_count}$; $P /= \text{sum}(P)$ // normalize distribution P

$\text{candidate_distribution_Q} = \text{quantize}[\text{bin}[0], \dots, \text{bin}[i-1]]$ into 128 levels

 expand $\text{candidate_distribution_Q}$ to 'i' bins

$Q /= \text{sum}(Q)$ // normalize distribution Q

$\text{divergence}[i] = \text{KL_divergence}(\text{reference_distribution_P}, \text{candidate_distribution_Q})$

End For

- Extra treatment for 0-element bins

Example: quantize P into 2 bins

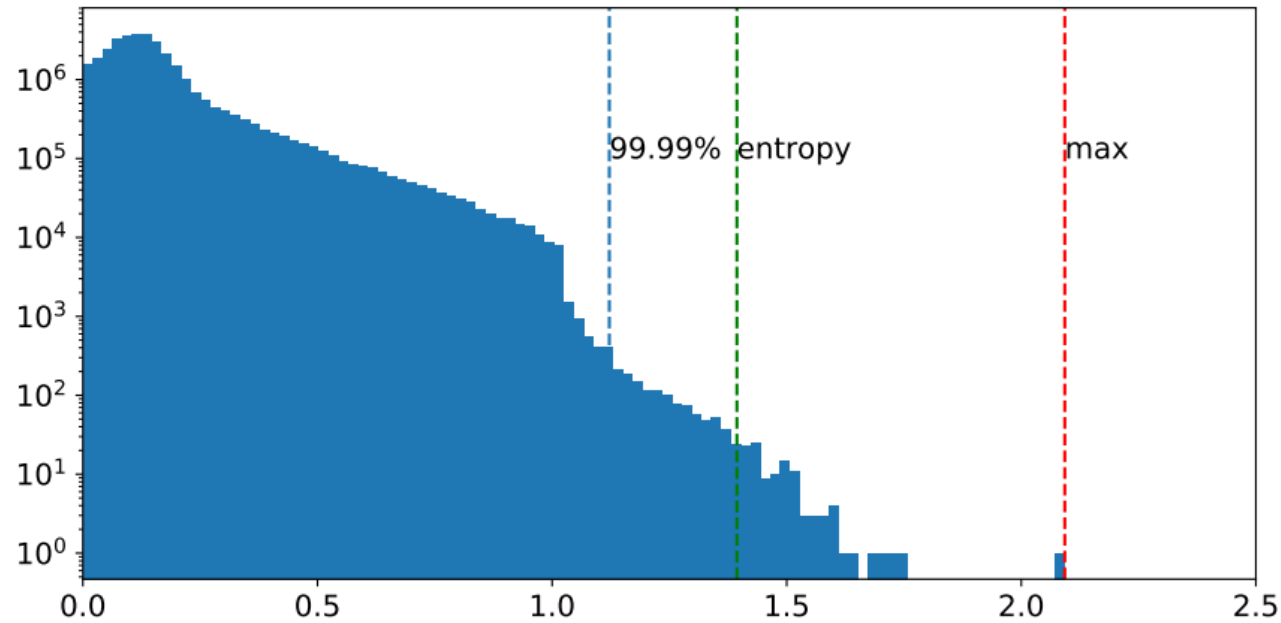
$P = [1, 0, 2, 3, 5, 3, 1, 7] \rightarrow [1+0+2+3, 5+3+1+7]$

Expand and reserve zero bins

$Q = [2, 0, 2, 2, 4, 4, 4, 4]$



Comparison of Different Strategies



Histogram of input activations to layer 3 in ResNet50 and calibrated ranges

Other Optimizations to Consider

- Rounding, AdaRound (Markus Nagel, 2020)

$$\arg \min_{\text{rounding}} |XW - \hat{X}\hat{W}|^2$$



$$\arg \min_{\text{rounding}} |L(X, W + \Delta W) - L(X, W)|$$



Taylor expansion

$$|L(X, W + \Delta W) - L(X, W)| = \left| \underbrace{\nabla L(X, W) \Delta W}_0 + \frac{1}{2} \Delta W^T \nabla^2 L(X, W) \Delta W \right|$$



$$\arg \min_{\Delta W} \left| \frac{1}{2} \Delta W^T \nabla^2 L(X, W) \Delta W \right|$$



AdaRound

- Rounding-to-nearest with $H = \nabla^2 L(X, W) = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$ $\arg \min_{\Delta W} \left| \frac{1}{2} \Delta W^T H \Delta W \right|$

$$\Delta W^T H \Delta W = \Delta w_1^2 + \Delta w_2^2 + \Delta w_1 \Delta w_2$$

- Hessian approximation for a FFN $z = WX$

$$H = \frac{\partial^2 L}{\partial w^2} = \frac{X^T \frac{\partial L}{\partial z}}{\partial w} = \frac{X^T \partial g}{\partial w} = X^T \frac{\partial g}{\partial z} \frac{\partial z}{\partial w} = X^T X \frac{\partial^2 L}{\partial z^2}$$

Approximated by an identity matrix

$$\arg \min_{\Delta W} \left| \frac{1}{2} \Delta W^T H \Delta W \right| \quad \Rightarrow \quad \arg \min_{\Delta W} |(X \Delta W)^2|$$

Classification problem



AdaRound (Cont'd)

- Loss function for rounding

$$\hat{W} = s \text{ clip} \left(\left\lfloor \frac{W}{s} \right\rfloor + h(V_{i,j}), n, p \right)$$

$$h(\alpha_{i,j}) = \text{clip}(\sigma(V_{i,j}) \times 1.2 - 0.1, 0, 1)$$

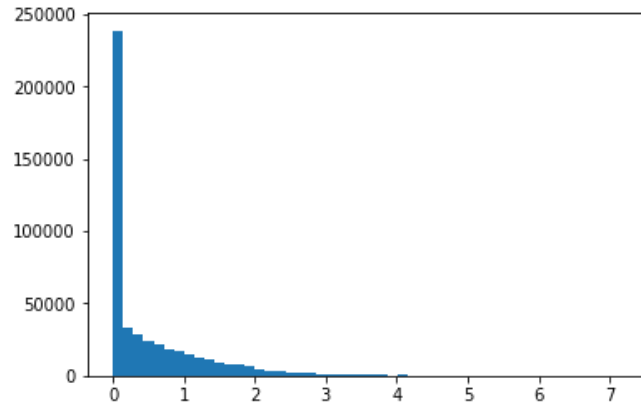
$$\arg \min_{\Delta W} |(\mathbf{X}\Delta W)^2| + \lambda \sum_{i,j} (1 - |2h(V_{i,j}) - 1|^\beta)$$

- Use Adam optimizer to optimize $V_{i,j}$
- Quantize ResNet50 to 4-bit weights

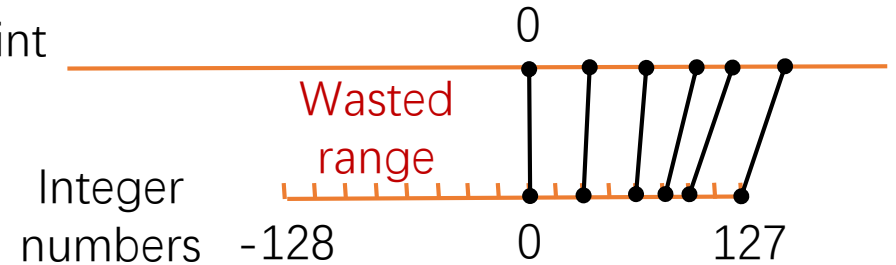


Other Optimizations to Consider (Cont'd)

- What if quantize ReLU results to signed integers?



Floating-point values



- Asymmetric quantization, still linear

$$x_{int} = \text{clamp} \left[\text{round} \left(\frac{x}{s_x} + z \right); 0, 2^{BW} - 1 \right] \text{ (unsigned integer)}$$

or

$$x_{int} = \text{clamp} \left[\text{round} \left(\frac{x}{s_x} + z \right); -2^{BW-1}, 2^{BW-1} - 1 \right] \text{ (signed integer)}$$



Dequantization: $\hat{x} = (x_{int} - z) * s_x$

z is preferably an integer.

Quantization in Neural Network

- Quantization has to work with the layer-by-layer structure of neural network

$$X = \begin{pmatrix} 0.5 & 0.7 \\ 0.4 & 0.5 \end{pmatrix} \quad W = \begin{pmatrix} 0.3 & 0.3 \\ 0.3 & 0.9 \end{pmatrix}$$



Quantize with
scaling factor 1/10

$$X_{int} = \begin{pmatrix} 5 & 7 \\ 4 & 5 \end{pmatrix} \quad W_{int} = \begin{pmatrix} 3 & 3 \\ 3 & 9 \end{pmatrix}$$

$$V = XW = \begin{pmatrix} 0.36 & 0.78 \\ 0.27 & 0.57 \end{pmatrix}$$



Re-quantize for next
layer computation



Dequantize with scaling factor
 $s_X s_W = 1/100$

$$V_{int} = X_{int} W_{int} = \begin{pmatrix} 36 & 78 \\ 27 & 57 \end{pmatrix}$$

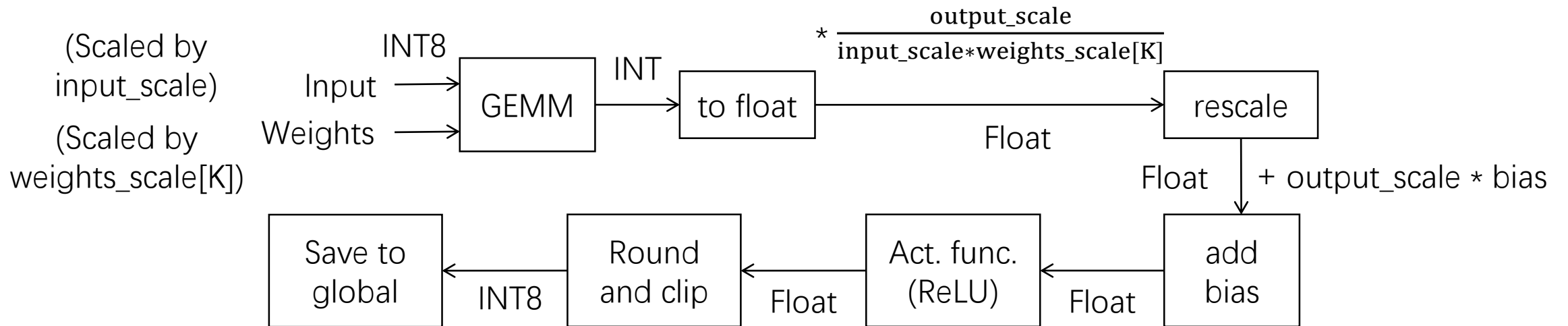


Consideration for Multi-layer NNs

- Quantization has to work with the layer-by-layer structure of neural network
- NVIDIA® TensorRT™ is an SDK that facilitates high performance machine learning inference. It focuses specifically on running an already-trained network quickly and efficiently on NVIDIA hardware [1].

- INT8_input, INT8_weights [2]; Float32_bias (original bias from the F32 model)
- Float32 scaling factors: input_scale, output_scale, weights_scale[K] (K-th channel)

Quantization
granularity

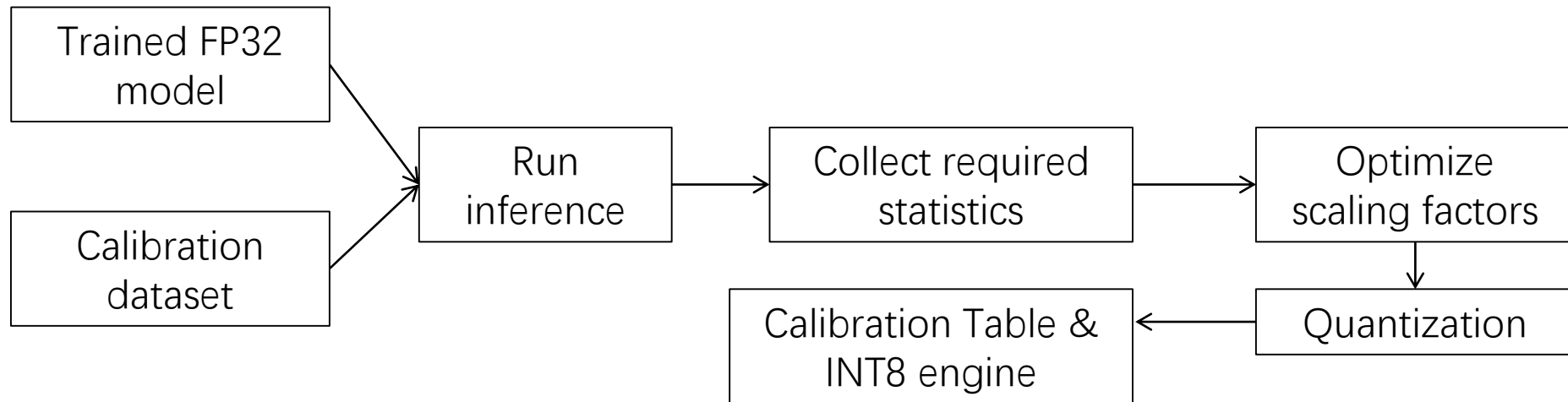


[1] NVIDIA TensorRT documentation. <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#intro-quantization>

[2] Szymon Migacz, NVIDIA. 8-bit inference with TensorRT. May 8th, 2017

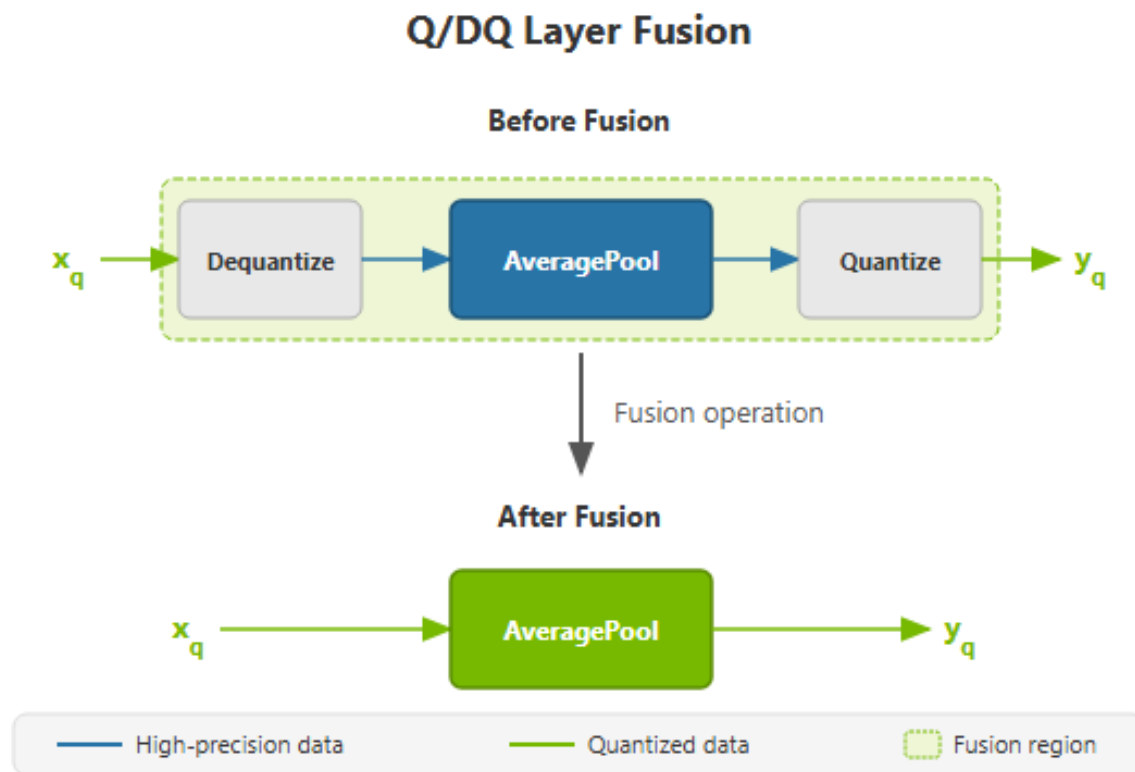
Quantization in TensorRT

- Automated, parameterless method, converting FP32 CNN models into INT8
- Symmetric, linear quantization for weights and acts./fmaps.
- Minimize information loss by default (entropy-based method)



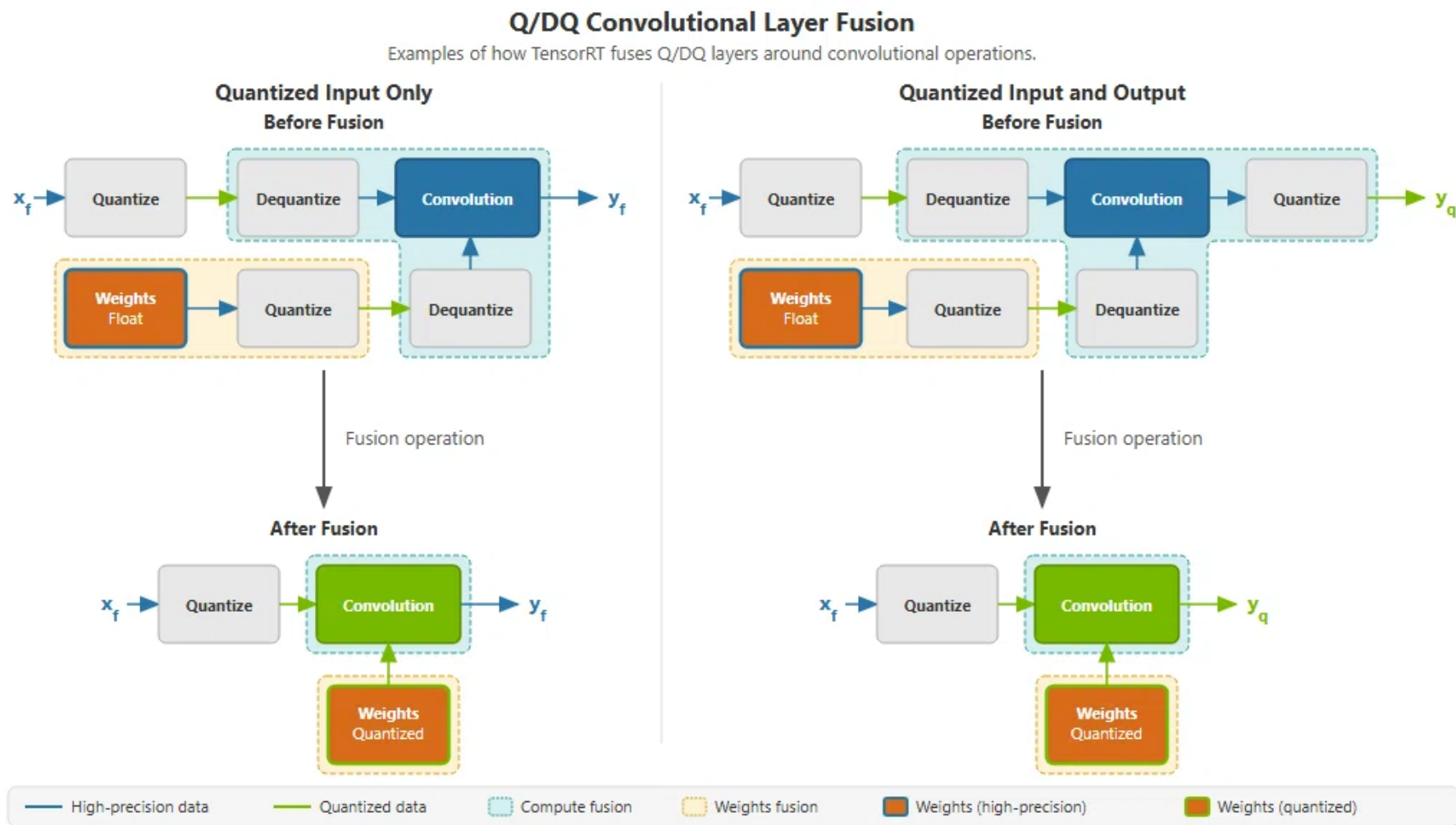
TensorRT Optimizes the Computation Graph

- Maximize the proportion of the graph that can be processed at low precision;
 - Push “Quantize” node backward
 - Pull “Dequantize” node forward
- Example



TensorRT Optimize the Computation Graph

- Example



[1] NVIDIA TensorRT documentation. <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#intro-quantization>

Alternatively

- Google's solution (Tensorflow Lite/TPU)

$$\begin{array}{ccc} \text{Real number} & \xleftarrow{\text{orange}} & r \approx S(q - Z) \\ & & \downarrow \text{orange} \\ & & \text{Quantized number} \end{array} \quad S > 0 \text{ (float)}$$

1. Matrix multiplication

$$\begin{array}{c} r_3 = r_1 r_2 \\ \downarrow \\ S_3(q_3 - Z_3) \approx S_1(q_1 - Z_1)S_2(q_2 - Z_2) \\ \downarrow \\ q_3 \approx Z_3 + M(q_1 - Z_1)(q_2 - Z_2) \\ 0 < M = \frac{S_1 S_2}{S_3} < 1 \end{array}$$

2. Represent M as

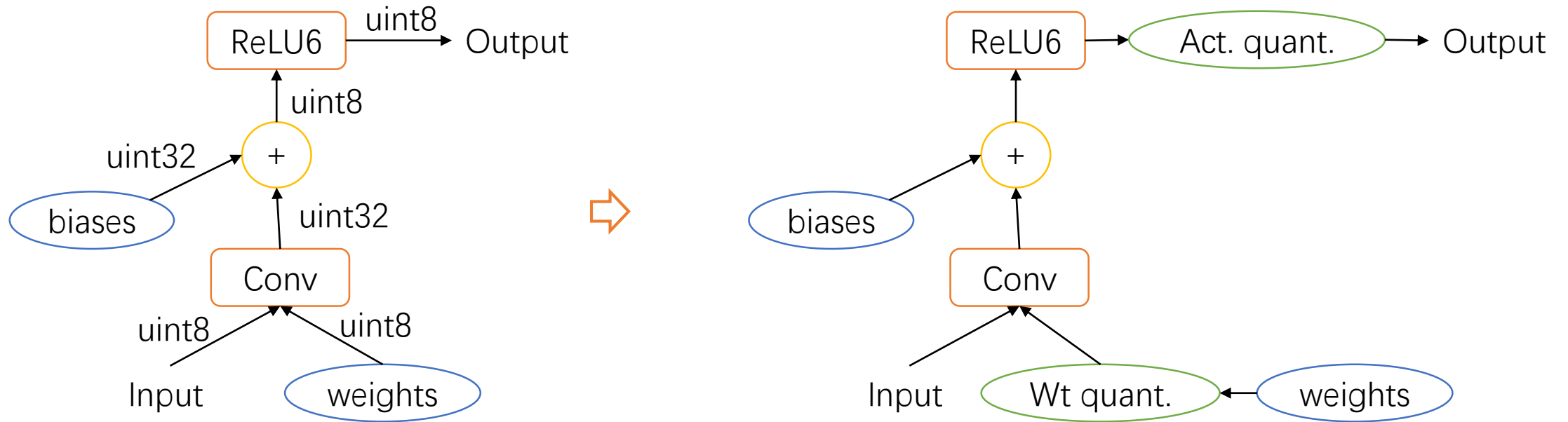
$$\begin{array}{ccc} & M = 2^{-n} M_0 & \\ \swarrow & & \searrow \\ \text{Bit shift} & & 0.5 \leq M_0 < 1 \\ & & \downarrow \\ & & \text{Fixed-point number} \end{array}$$

3. Considering bias term r_{bias} : q_{bias} **int32**

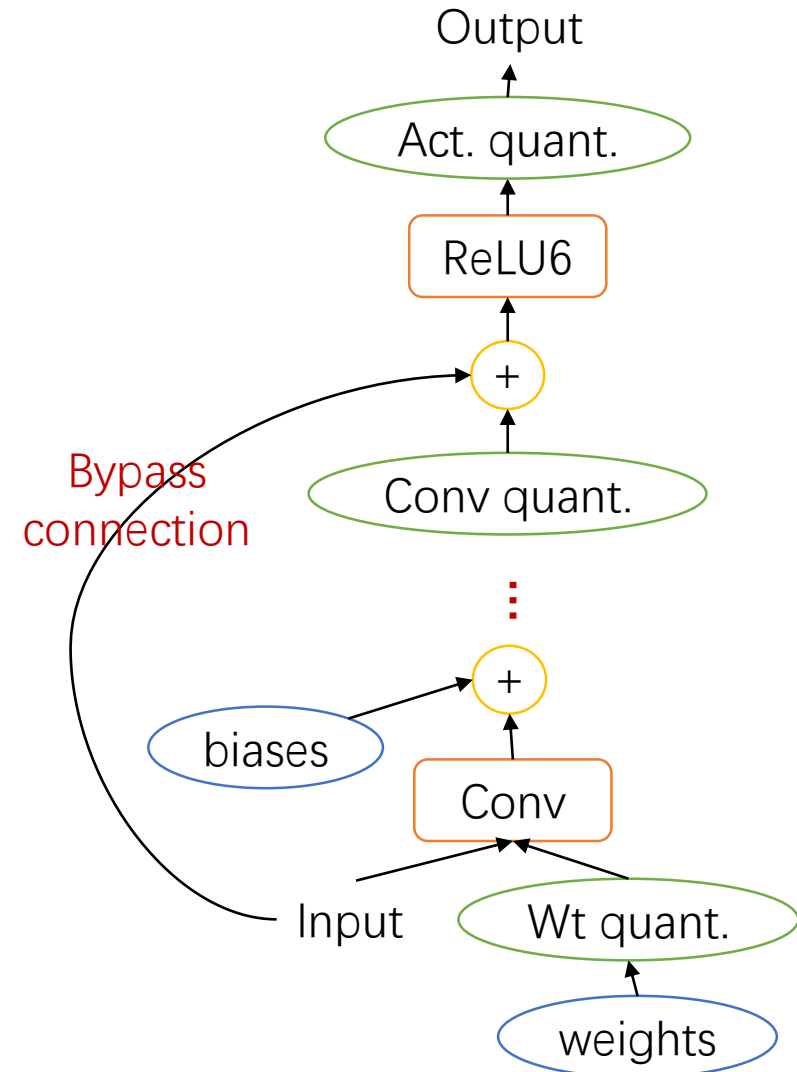
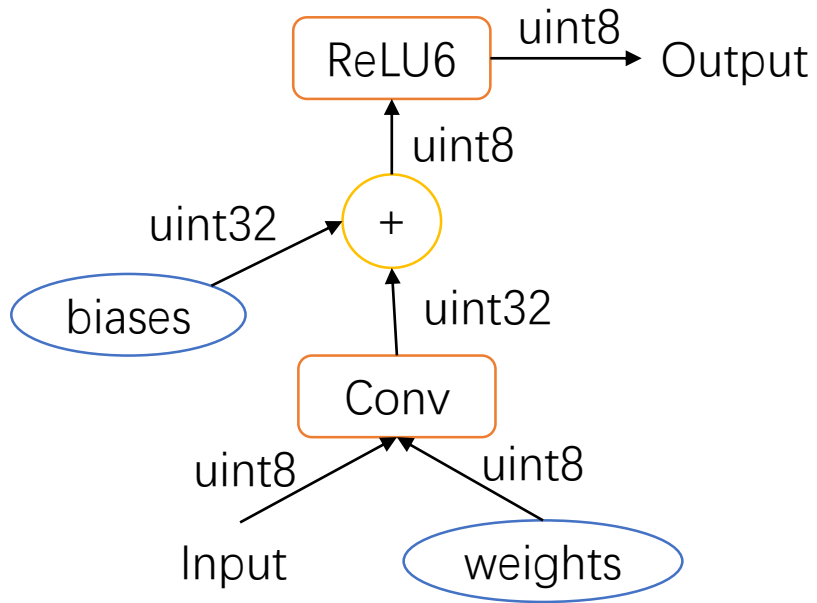
$$r_3 = r_1 r_2 + r_{bias}$$

$$\begin{array}{l} q_3 \approx Z_3 + M(q_1 - Z_1)(q_2 - Z_2) + M q_{bias} \\ S_{bias} = S_1 S_2, Z_{bias} = 0 \end{array}$$

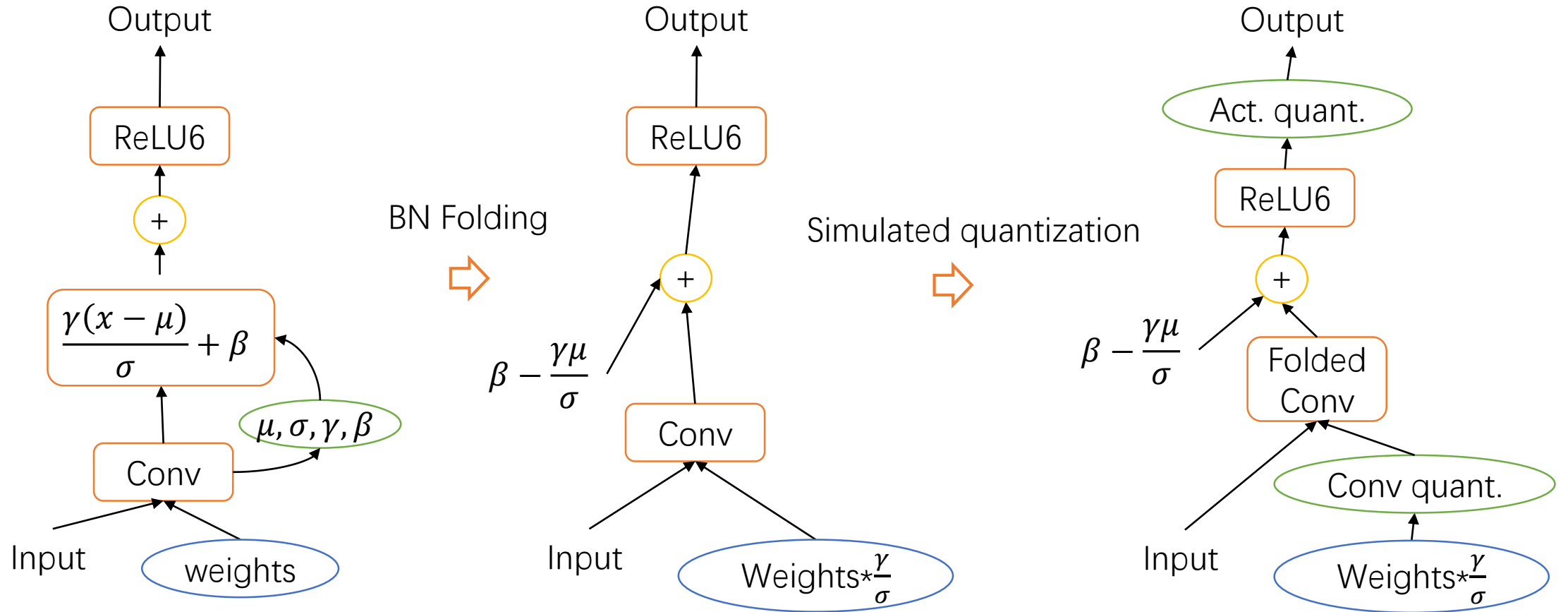
Simulated Quantization Computational Graph



Bypass Connections



BN Folding



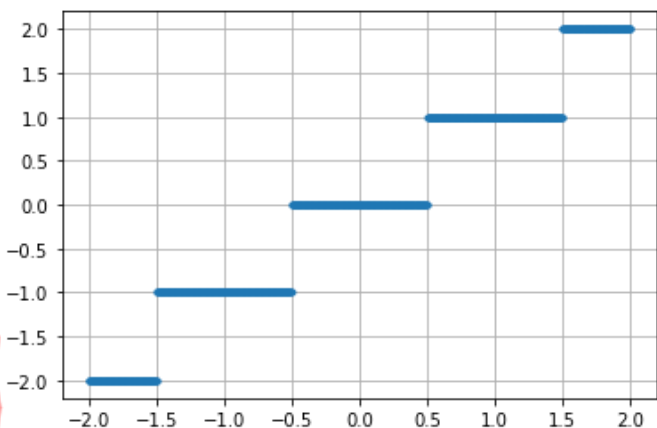
Quantization-aware Training (QAT)

- PTQ has limitations, especially when aiming for low-bit quantization, e.g., 4-bit and below

- **Simulated quantization & QAT**

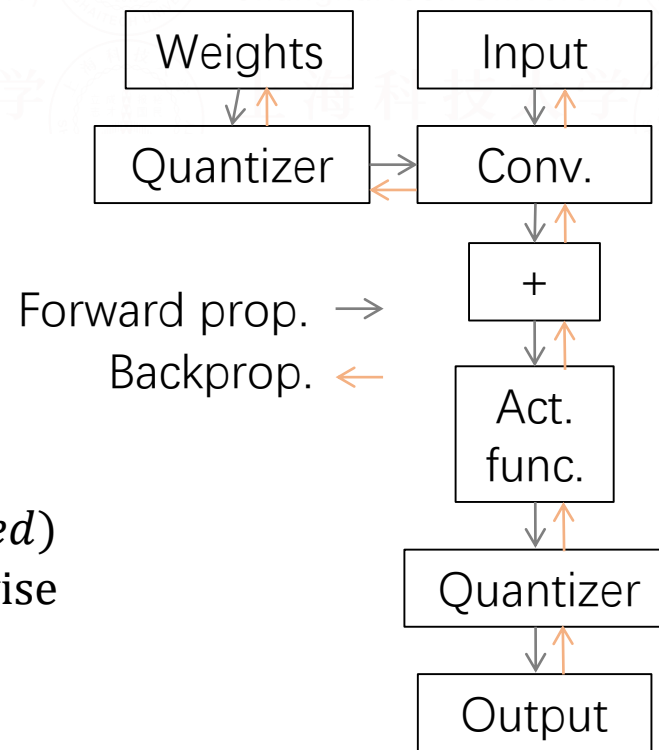
- Straight-through estimator (STE, backprop through the simulated quantizer block)

- $y = \text{round}(x); \frac{\partial y}{\partial x} \approx 1$



$$\begin{aligned} x &\approx \hat{x} = s_x(x_{int}) \\ &= s_x \left\{ \text{clamp} \left[\text{round} \left(\frac{x}{s_x} \right); 0, 2^{BW} - 1 \right] \right\} \end{aligned}$$

$$\frac{\partial \hat{x}}{\partial x} \approx \begin{cases} 1, & \text{if } 0 \leq x_{int} \leq 2^{BW} - 1 \text{ (unsigned)} \\ 0, & \text{otherwise} \end{cases}$$



QAT (Cont'd)

- Parameterized clipping activation (PACT)

$$y = \text{PACT}(x) \begin{cases} 0 & x \in (-\infty, 0) \\ x & x \in [0, \alpha) \\ \alpha & x \in [\alpha, +\infty) \end{cases}$$

$$\frac{\partial y}{\partial \alpha} = \begin{cases} 0 & x \in (-\infty, 0) \\ 1 & x \in [\alpha, +\infty) \end{cases}$$

- Adjust α only when clipping is activated

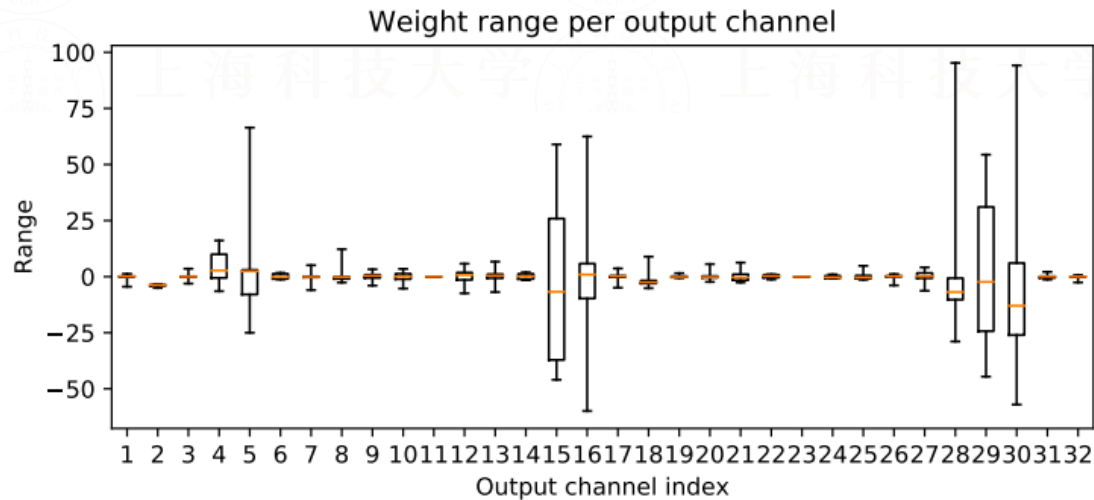
- Learned step size quantization (LSQ)
 - Learn scaling factor with training

$$\mathbf{x} \approx \hat{\mathbf{x}} = s_x(\mathbf{x}_{int}) = s_x \left\{ \text{clamp} \left[\text{round} \left(\frac{\mathbf{x}}{s_x} \right); 0, 2^{BW} - 1 \right] \right\}$$

$$\frac{\partial \hat{\mathbf{x}}}{\partial s_x} = \begin{cases} -\frac{\mathbf{x}}{s_x} + \text{round} \left(\frac{\mathbf{x}}{s_x} \right) & \frac{\mathbf{x}}{s_x} \text{ within bounds} \\ \text{lower bound} & \frac{\mathbf{x}}{s_x} < \text{lower bound} \\ \text{upper bound} & \frac{\mathbf{x}}{s_x} > \text{upper bound} \end{cases}$$

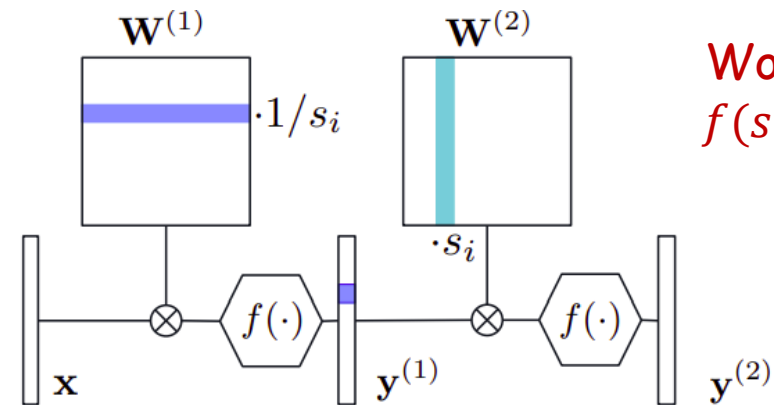
Other Quantization Methods

- Power of 2 quantization ($s_x = 2^k, k = \dots -2, -1, 1, 2, \dots$)
 - Allow easy scaling with bit-shifting s_x
- Depth-wise separable layers



One depthwise separable layer in MobileNet

- Solution 1
 - Per-channel quantization
- Solution 2
 - Cross-layer equalization (CLE) with per-tensor quant.

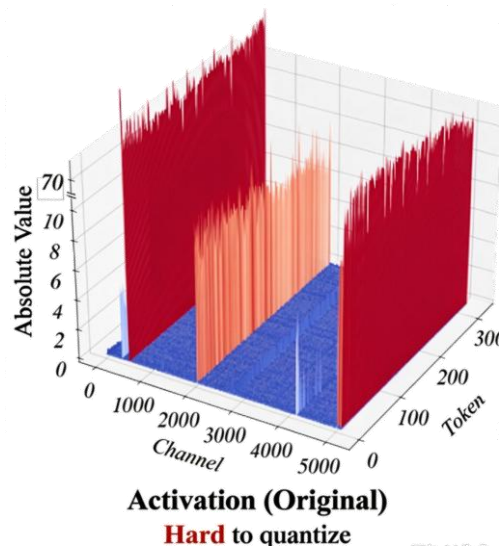
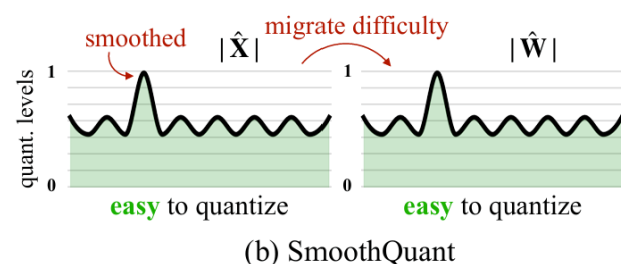
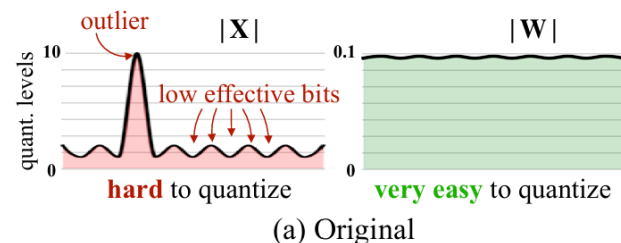


Works only when
 $f(s \cdot x) = s \cdot f(x)$,
e.g., ReLU

PTQ of LLMs: SmoothQuant

- Migrate scale variant from activations (outliers) to weights offline for ease of activation quantization
- Intuition
- Observation
- Solution

Only a few channels of activation shows large magnitudes



Example

$\mathbf{X}$	$\mathbf{W}$	$\max(\mathbf{W}_j)$
1	2	-2
-16	1	-1
2	-2	-2
6	-1	1
-2	-1	-2
8	-1	-2
-1	1	-2
-9	-1	-2
2	-1	-2
16	-1	-2
2	1	-2
9	-1	-2

Abs Max ↓

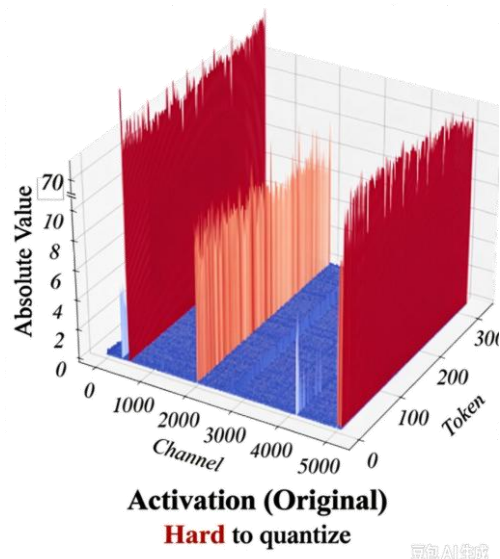
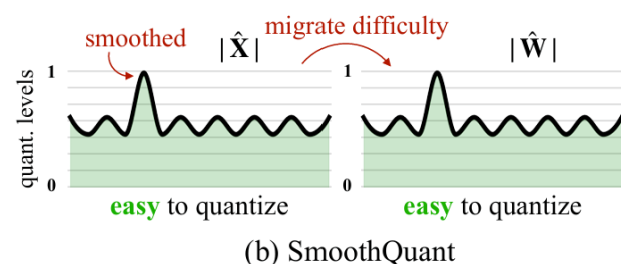
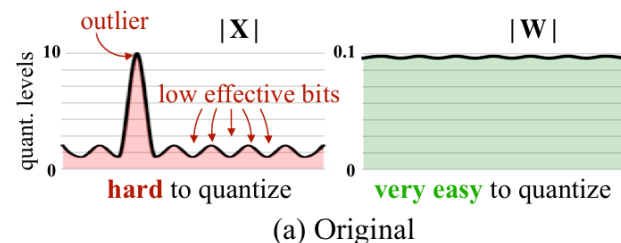
$\max(|\mathbf{X}_j|)$

$$\alpha = 0.5, \mathbf{s} = \sqrt{\max(|\mathbf{X}_j|) / \max(|\mathbf{W}_j|)} = [1, 4, 1, 3]$$

PTQ of LLMs: SmoothQuant

- Migrate scale variant from activations (outliers) to weights offline for ease of activation quantization
- Intuition
- Observation
- Solution

Only a few channels of activation shows large magnitudes



$$\mathbf{Y} = (\mathbf{X} \text{diag}(\mathbf{s})^{-1}) \cdot (\text{diag}(\mathbf{s}) \mathbf{W}) = \hat{\mathbf{X}} \hat{\mathbf{W}}$$

$$\mathbf{s}_j = \max(|\mathbf{X}_j|)^\alpha / \max(|\mathbf{W}_j|)^{1-\alpha}$$

SmoothQuant

$$\hat{\mathbf{X}}$$

1	-4	2	2
-2	2	-1	-3

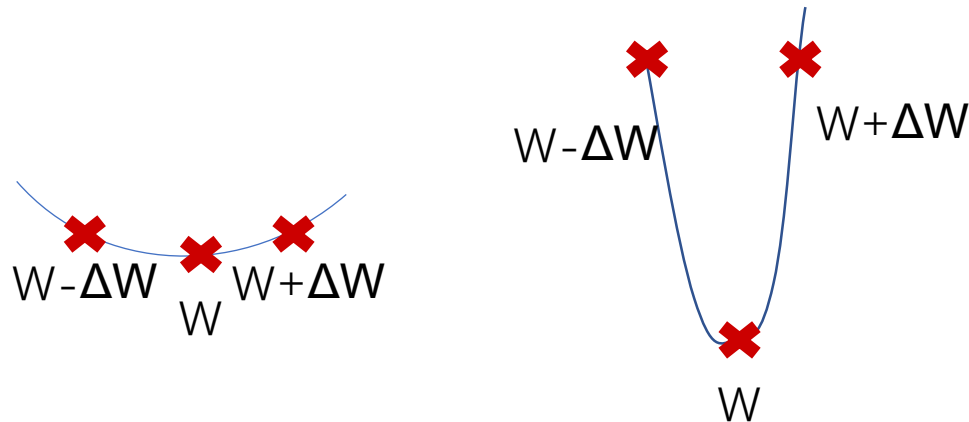
$$\hat{\mathbf{W}}$$

2	1	-2
4	-4	-4
2	-1	-2
-1	-1	1

$$\alpha = 0.5, \mathbf{s} = \sqrt{\max(|\mathbf{X}_j|) / \max(|\mathbf{W}_j|)} = [1, 4, 1, 3]$$

Other Quantization Methods (Cont'd)

- Homogeneous bit-width vs. mixed-precision
 - Flat loss landscape: lower bit precision required;
 - Sharp loss landscape: higher bit precision required
 - Using Hessian or top Hessian eigenvalue to guide the choice of bit precision layer by layer
 - Further optimize the trade-offs with integer linear programming [1]



$$\frac{\partial(g_i^T v)}{\partial W_i} = \frac{\partial g_i^T}{\partial W_i} v + g_i^T \frac{\partial v}{\partial W_i} = \frac{\partial g_i^T}{\partial W_i} v = H_i v$$

Algorithm 1: Power Iteration for Eigenvalue Computation

Input: Block Parameter: W_i .

Compute the gradient of W_i by backpropagation, i.e.,

$$g_i = \frac{dL}{dW_i}.$$

Draw a random vector v (same dimension as W_i).

Normalize v , $v = \frac{v}{\|v\|_2}$

for $i = 1, 2, \dots, n$ **do** // Power Iteration

 Compute $gv = g_i^T v$ // Inner product

 Compute Hv by backpropagation, $Hv = \frac{d(gv)}{dW_i}$

 // Get Hessian vector product

 Normalize and reset v , $v = \frac{Hv}{\|Hv\|_2}$

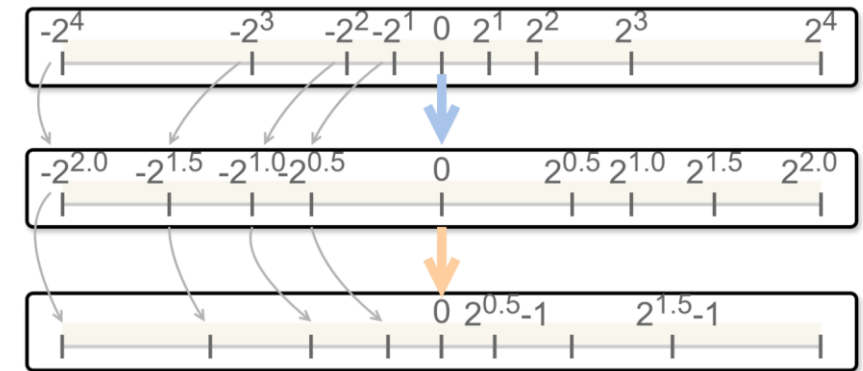
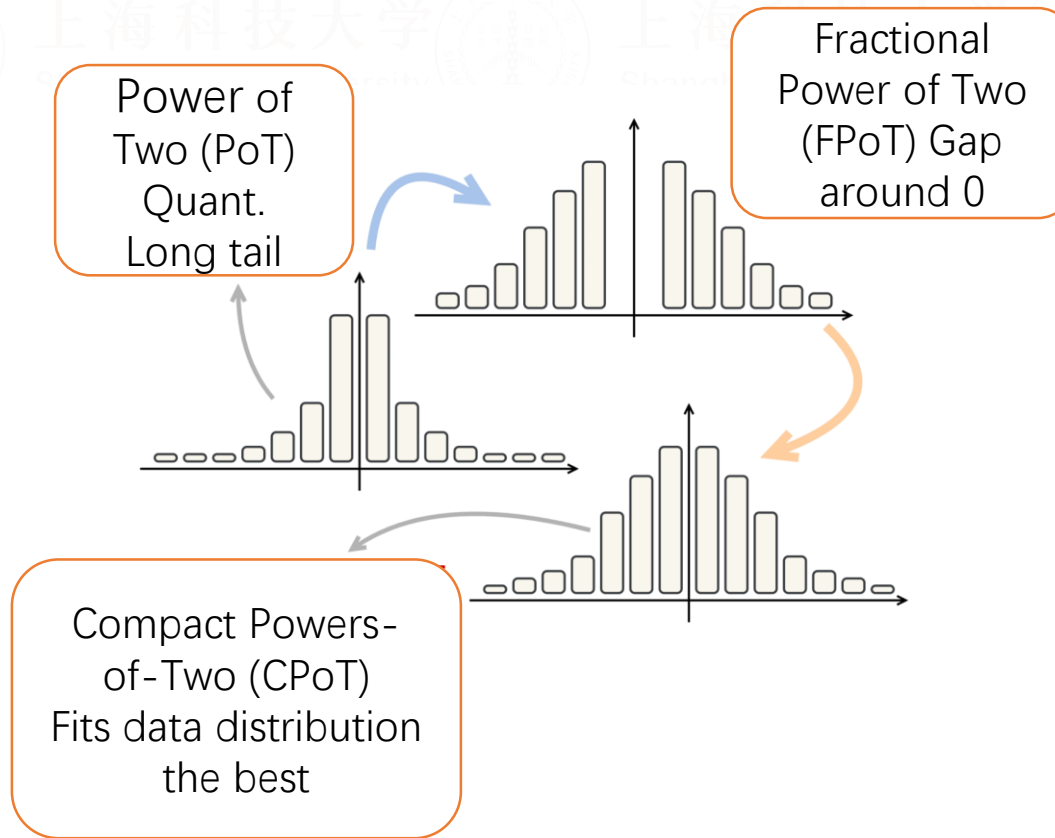
[1] HAWQ: Hessian AWARE Quantization of Neural Networks with Mixed-Precision.

[2] Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT.

[3] I-BERT: Integer-only BERT Quantization. [4] HAWQ-V3: Dyadic Neural Network Quantization

Other Quantization Methods (Cont'd)

- Non-uniform (non-linear) quantization
 - For example, quantization with logarithmic number system

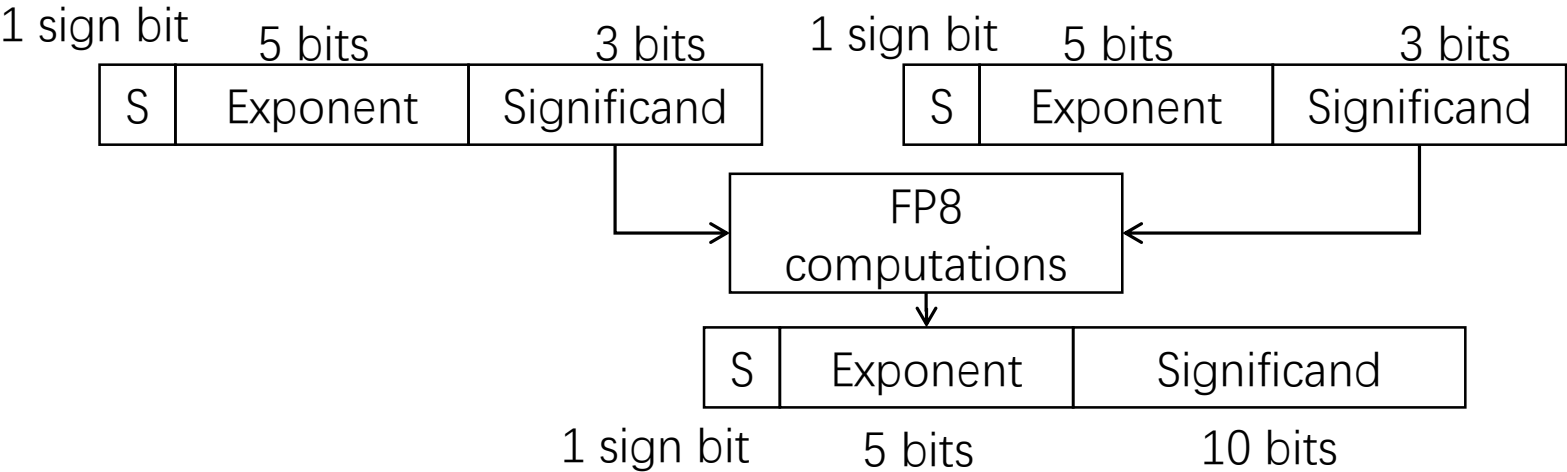


Recent Developments

- Low-precision floating-point format: bf16/FP8/FP4

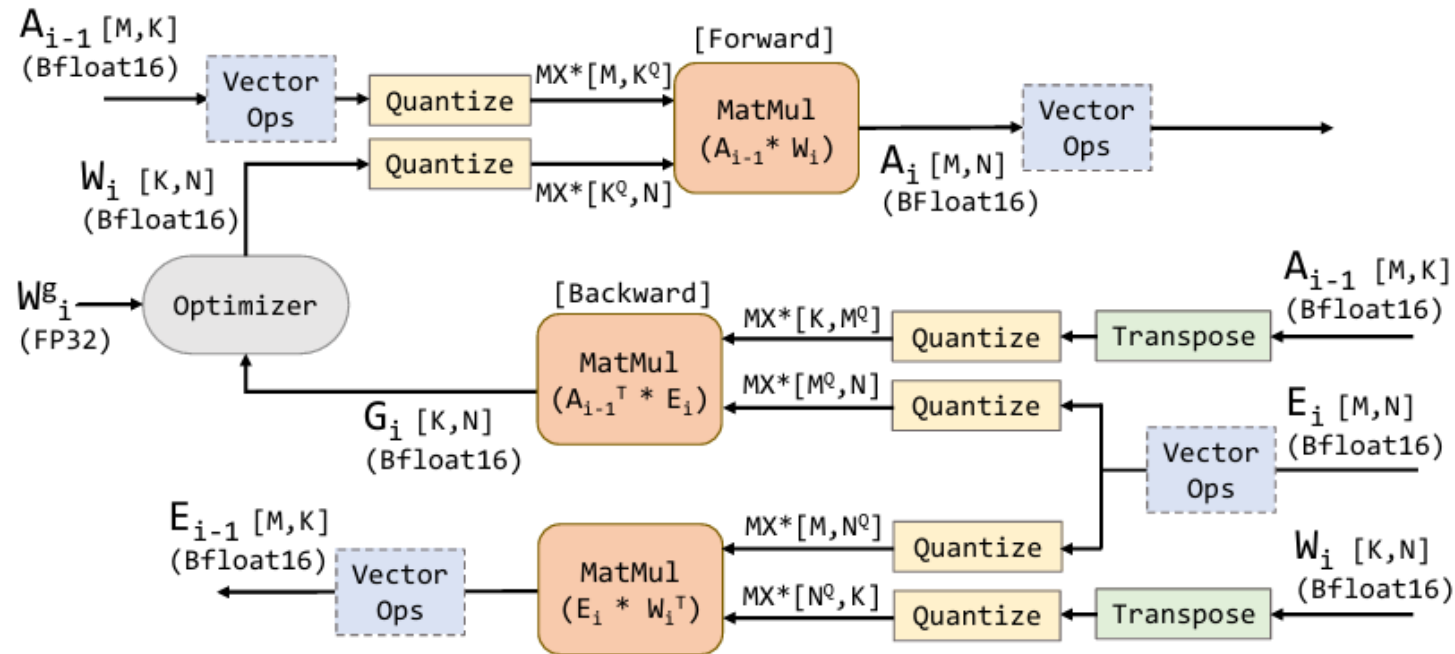
	E4M3	E5M2
Exponent bias	7	15
Infinities	N/A	$S.11111.00_2$
NaN	$S.1111.111_2$	$S.11111.\{01, 10, 11\}_2$
Zeros	$S.0000.000_2$	$S.00000.00_2$
Max normal	$S.1111.110_2 = 1.75 * 2^8 = 448$	$S.11110.11_2 = 1.75 * 2^{15} = 57,344$
Min normal	$S.0001.000_2 = 2^{-6}$	$S.00001.00_2 = 2^{-14}$
Max subnorm	$S.0000.111_2 = 0.875 * 2^{-6}$	$S.00000.11_2 = 0.75 * 2^{-14}$
Min subnorm	$S.0000.001_2 = 2^{-9}$	$S.00000.01_2 = 2^{-16}$

Model	Baseline	FP8
VGG-16	71.27	71.11
VGG-16 BN	73.95	73.69
Inception v3	77.23	77.06
DenseNet 121	75.59	75.33
DenseNet 169	76.97	76.83
Resnet18	70.58	70.12
Resnet34	73.84	73.72
Resnet50 v1.5	76.71	76.76
Resnet101 v1.5	77.51	77.48
ResNeXt50	77.68	77.62
Xception	79.46	79.17
MobileNet v2	71.65	71.04
DeiT small	80.08	80.02



Recent Developments (Cont'd)

- “Heterogeneous format”: micro-scaling
 - Divide tensor as 32-element blocks with shared scaling factor of powers of 2 (E8M0)
 - For each block, use either FP8/FP6/FP4/INT8 to represent a number
 - Recent NVFP4 is also a variant of micro-scaling



Model Compression Pruning and Beyond

Instructor: Siting Liu

School of Information Science and Technology (SIST), ShanghaiTech University

Shanghai Institute of Semiconductor Innovation (SISI)

<https://faculty.sist.shanghaitech.edu.cn/liust/EE219/EE219.html>



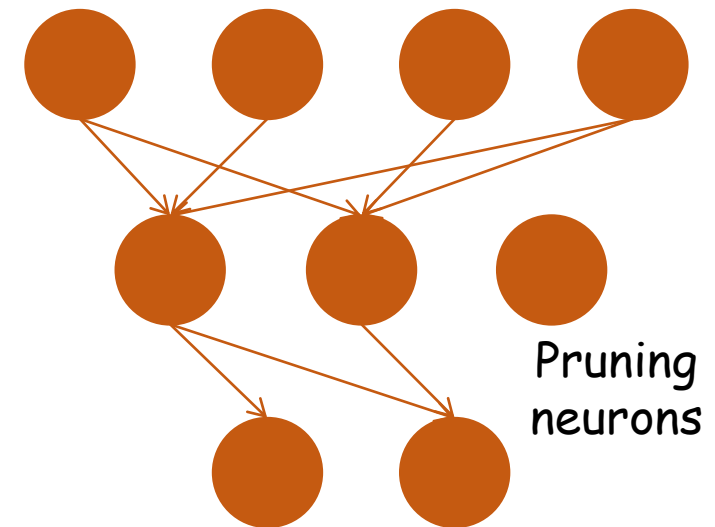
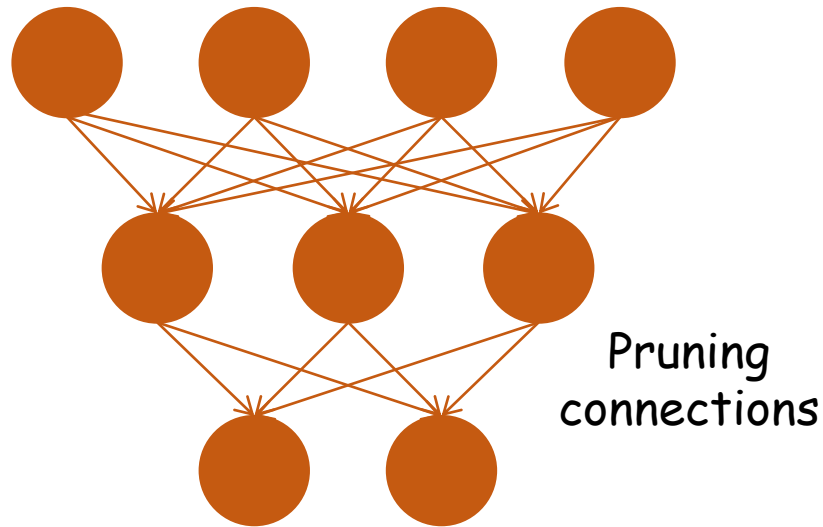
Computer Architecture - EE219
faculty.sist.shanghaitech.edu.cn

立志成才 报国裕民



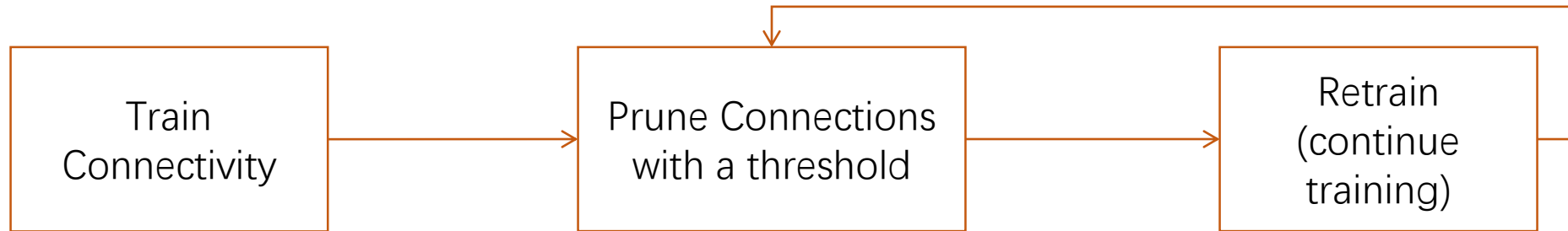
Model Compression: Pruning

- Some of the weights are near-zero and contribute “tiny” to the results;
- Reduce connections: reduce complexity and possibly over-fitting



Pruning: Magnitude-based Method

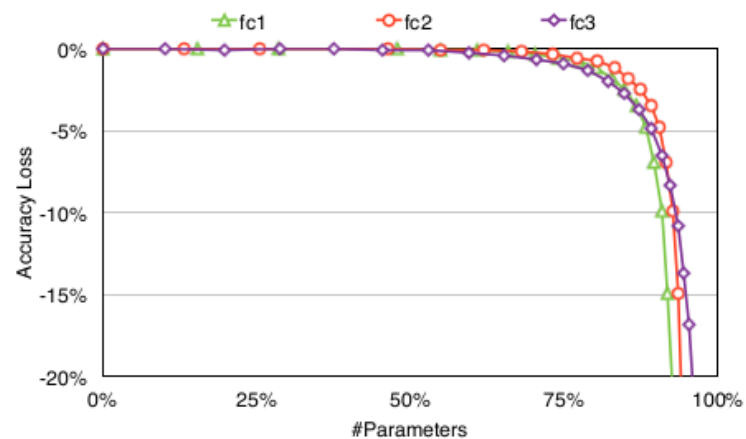
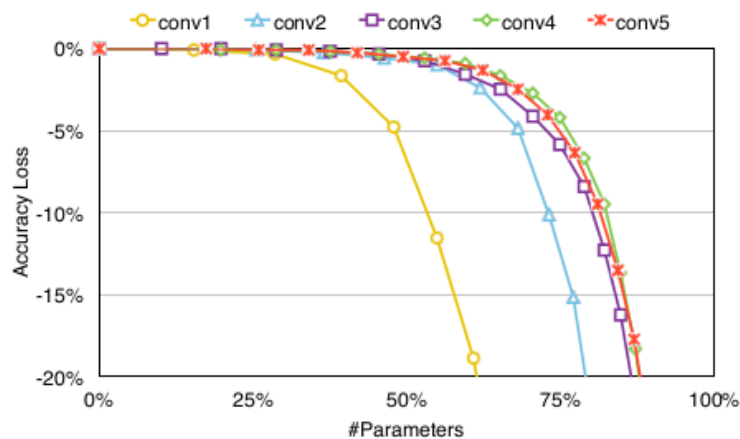
- Different from dropout (restore the connections after training)
- Reduce complexity and possibly over-fitting (somewhat equivalent to weight decay)



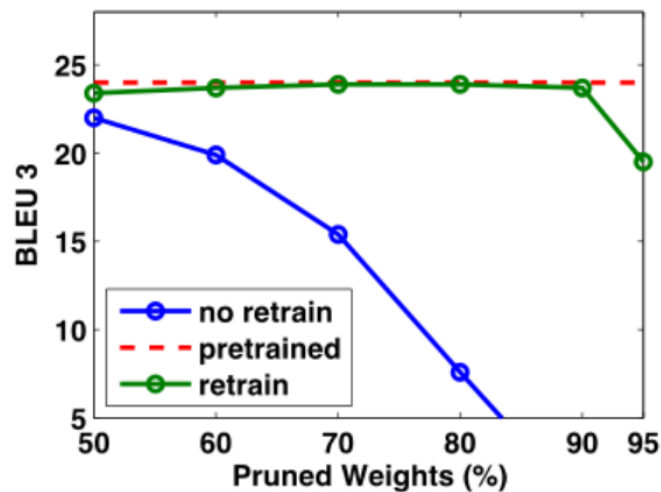
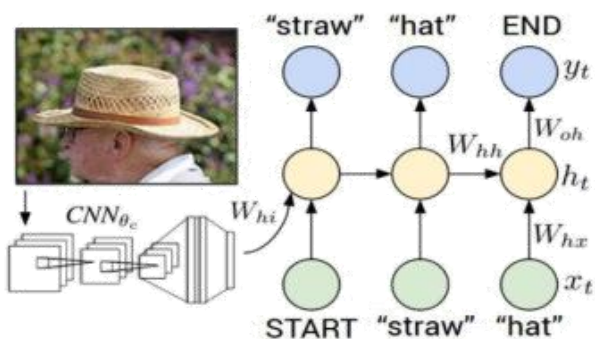
- Threshold chosen based on standard deviation of the layer weights
- Can also be specified with percentage or absolute number of pruned weights
- Retrain can be combined with dropout, but with smaller dropout ratio

Pruning on Classic NN Models

Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate	MNIST
LeNet-300-100 Ref	1.64%	-	267K		
LeNet-300-100 Pruned	1.59%	-	22K	12×	
LeNet-5 Ref	0.80%	-	431K		
LeNet-5 Pruned	0.77%	-	36K	12×	ImageNet
AlexNet Ref	42.78%	19.73%	61M		
AlexNet Pruned	42.77%	19.67%	6.7M	9×	
VGG-16 Ref	31.50%	11.32%	138M		
VGG-16 Pruned	31.34%	10.88%	10.3M	13×	



Pruning on RNN/LSTM



90%



- **Original:** a basketball player in a white uniform is playing with a **ball**
- **Pruned 90%:** a basketball player in a white uniform is playing with a **basketball**

90%



- **Original :** a brown dog is running through a grassy **field**
- **Pruned 90%:** a brown dog is running through a grassy **area**

90%



- **Original :** a man is riding a surfboard on a wave
- **Pruned 90%:** a man in a wetsuit is riding a wave **on a beach**

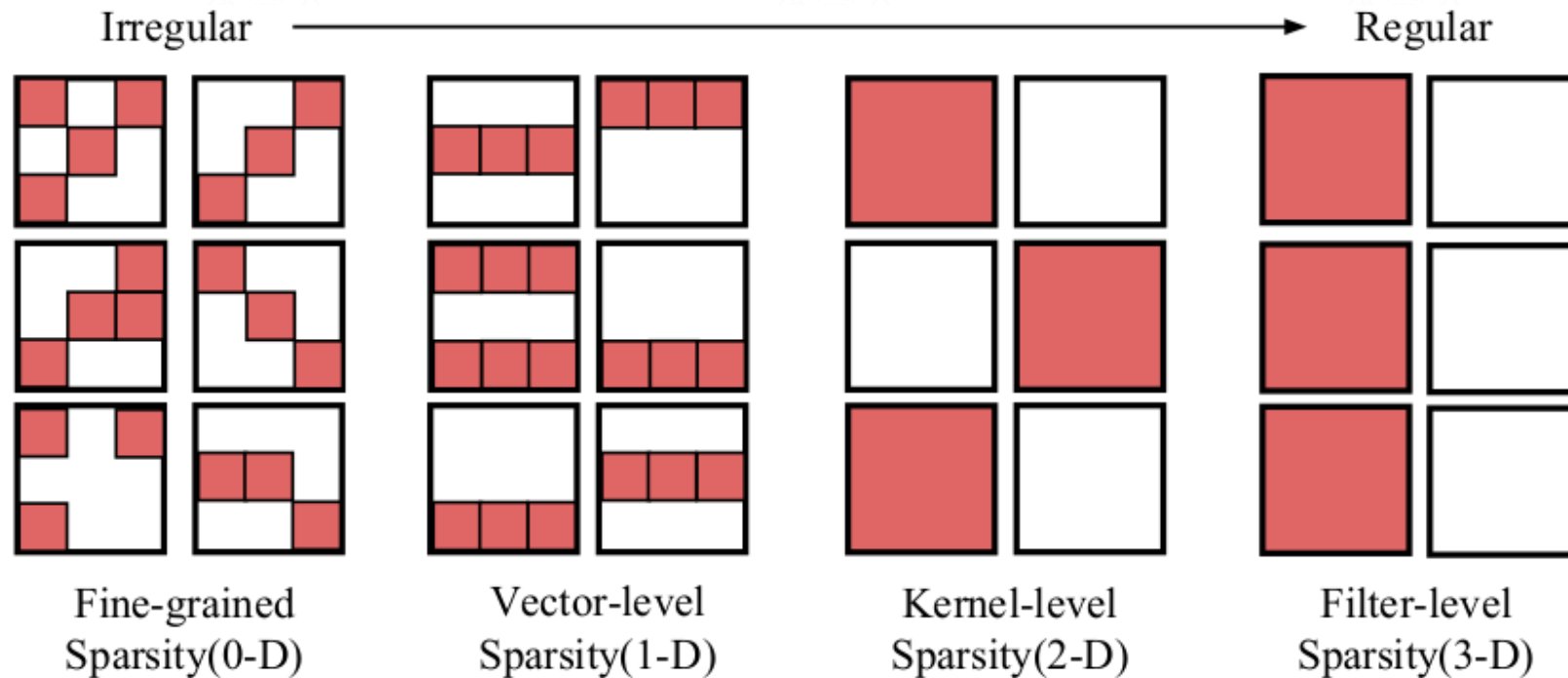
95%



- **Original :** a soccer player in red is running in the field
- **Pruned 95%:** a man in a **red shirt and black and white black shirt** is running through a field

Structured vs. Unstructured Pruning

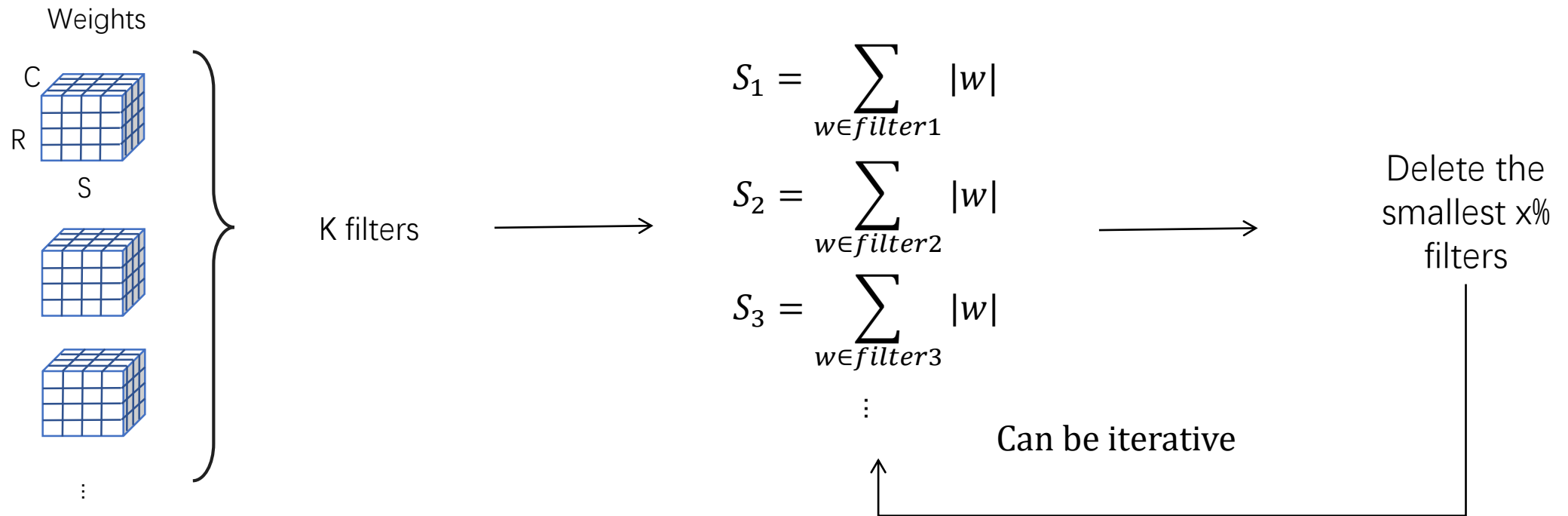
- Random pruning comes at the cost of the irregularity of the sparse computation pattern, poor utilization of parallel structure (cache, SIMD, etc.)
- Structured pruning, such as pruning entire filters will cause larger accuracy loss than pruning individual weights



Structured Pruning

- Pruning methods with different granularity;
- Groups with the smallest x% L1-norm are deleted

$$\text{Saliency: } S_i = \sum_{w \in G_i} |w|$$

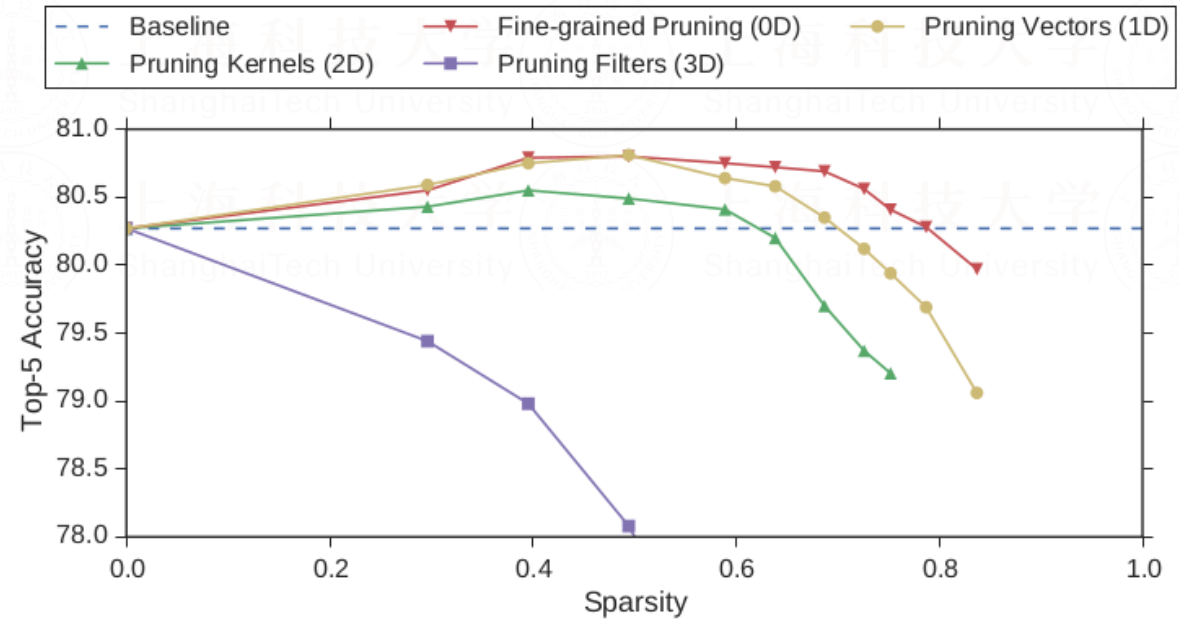
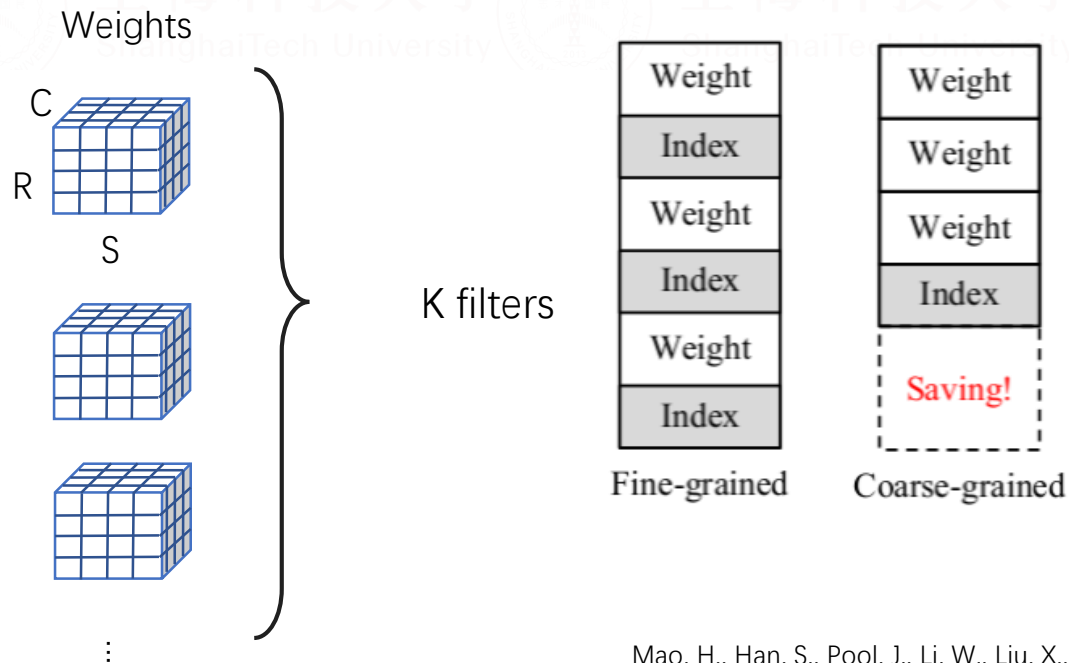


Mao, H., Han, S., Pool, J., Li, W., Liu, X., Wang, Y., & Dally, W. J. (2017). Exploring the regularity of sparse structure in convolutional neural networks.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. NIPS'16.

Structured Pruning (Cont'd)

- Fine-grained (0-Dim, unstructured)
- Vector (1-Dim) sub-kernel of size S
- Kernel (2-Dim) Kernels of shape $R \times S$
- Filter (3-Dim)

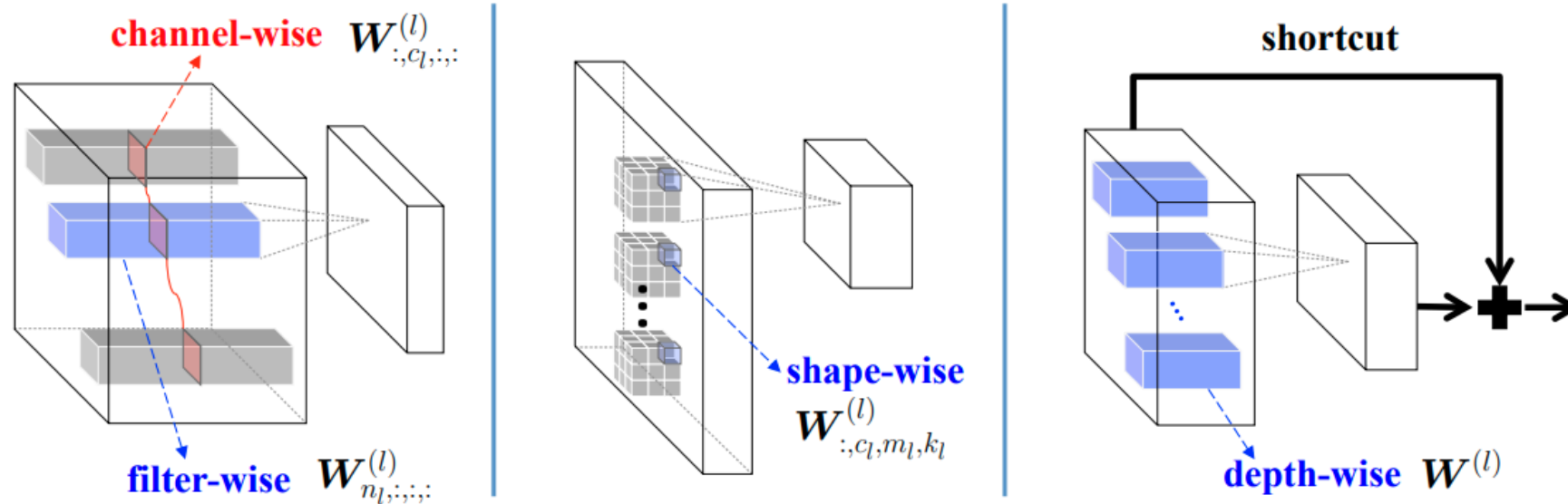


AlexNet on ImageNet dataset

Mao, H., Han, S., Pool, J., Li, W., Liu, X., Wang, Y., & Dally, W. J. (2017). Exploring the regularity of sparse structure in convolutional neural networks.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. NIPS'16.

Structured Pruning - Alternate Approach



$$E(\mathbf{W}) = E_D(\mathbf{W}) + \lambda_n \cdot \sum_{l=1}^L \left(\sum_{n_l=1}^{N_l} \|\mathbf{W}_{n_l,:}^{(l)}\|_g \right) + \lambda_c \cdot \sum_{l=1}^L \left(\sum_{c_l=1}^{C_l} \|\mathbf{W}_{:,c_l,:}^{(l)}\|_g \right)$$

Original cost function

The L th layer

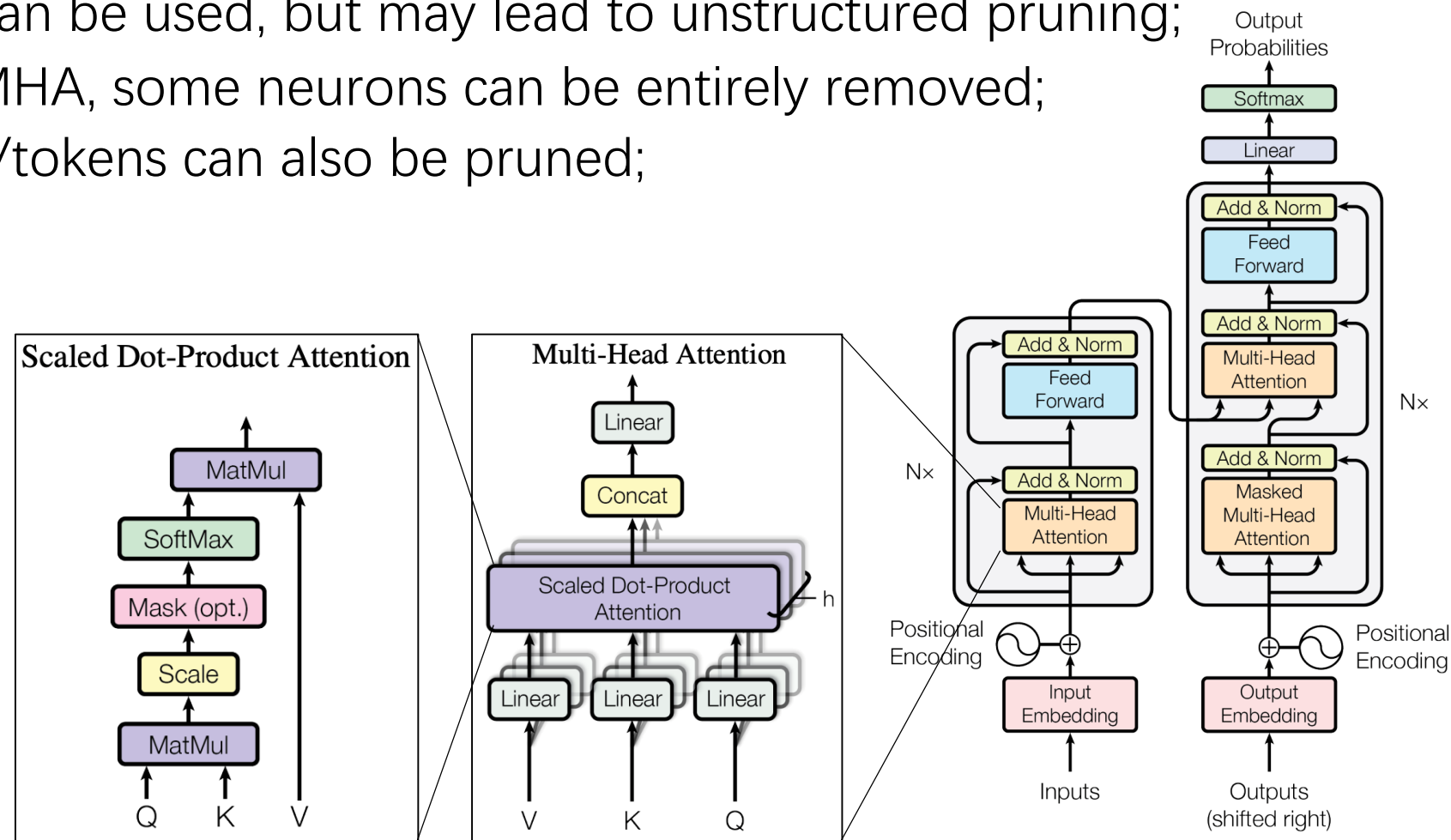
The n_l th filter

The L th layer

The c_l th channel

Pruning in Attention & LLM

- Similar pruning strategy can be used, but may lead to unstructured pruning;
 - In the FFN following MHA, some neurons can be entirely removed;
 - Wording embeddings/tokens can also be pruned;
- Head pruning
- Layer pruning
- Fine-tuning difficulty



Pruning Metric

- Magnitude-based method
- Loss-based method: the weight can be small, but the input can be huge!
 - Hessian-based approach, apply OBS to optimize.

$$L(W) - L(W^M) \approx (W - W^M)^T \nabla L(W^M) + \frac{1}{2} (W - W^M)^T H(W^M) (W - W^M)$$



Optimal Brain Surgeon (OBS)

- Prune and adjust the weights, so that the change of loss function is small

$$L(W) - L(\tilde{W}) \approx (W - \tilde{W})^T \nabla L(\tilde{W}) + \frac{1}{2} (W - \tilde{W})^T H(\tilde{W}) (W - \tilde{W})$$

- Assume weight q is pruned, i.e. $\delta w_q = -w_q$ or $e_q^T \delta W + w_q = 0$

- Apply Lagrange multiplier $\mathcal{L}(\delta w, \lambda) = \frac{1}{2} \delta w^T H \delta w + \lambda (e_q^T \delta w + w_q)$

- Let $\frac{\partial \mathcal{L}}{\partial \delta w} = H \delta w + \lambda e_q = 0$, then $\delta w = -\lambda H^{-1} e_q$

- $-\lambda e_q^T H^{-1} e_q + w_q = 0$, so that $\lambda = \frac{w_q}{[H^{-1}]_{qq}}$

- So $\delta w = -\frac{w_q}{[H^{-1}]_{qq}} H^{-1} e_q$ and $L_q = \frac{w_q^2}{2[H^{-1}]_{qq}}$

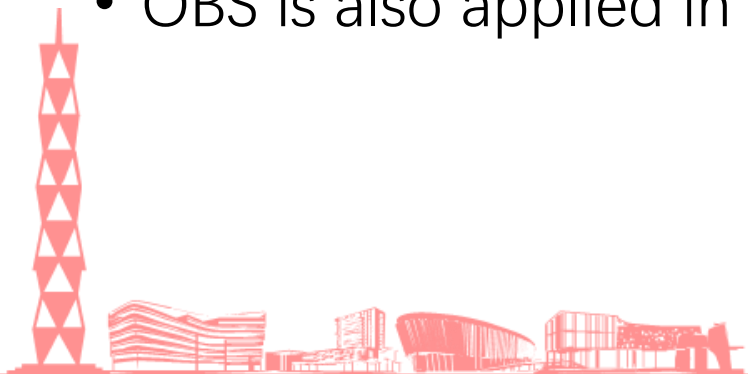


Pruning Metric (Cont'd)

- Magnitude-based method
- Loss-based method: the weight can be small, but the input can be huge!
 - Hessian-based approach, apply OBS to optimize.

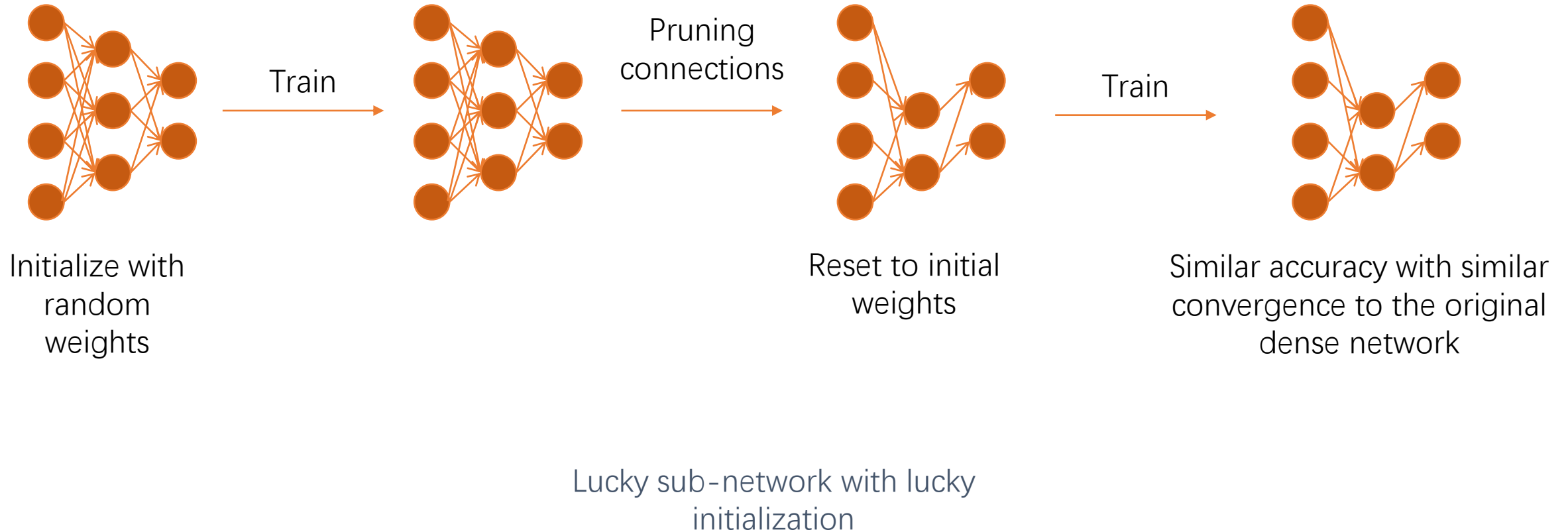
$$L(W) - L(W^M) \approx (W - W^M)^T \nabla L(W^M) + \frac{1}{2} (W - W^M)^T H(W^M) (W - W^M)$$

- SparseGPT uses OBS for column-wise fine pruning;
- SlimGPT uses OBS for structured pruning (attention heads/channels, etc.);
- OBS is also applied in quantization (GPTQ).



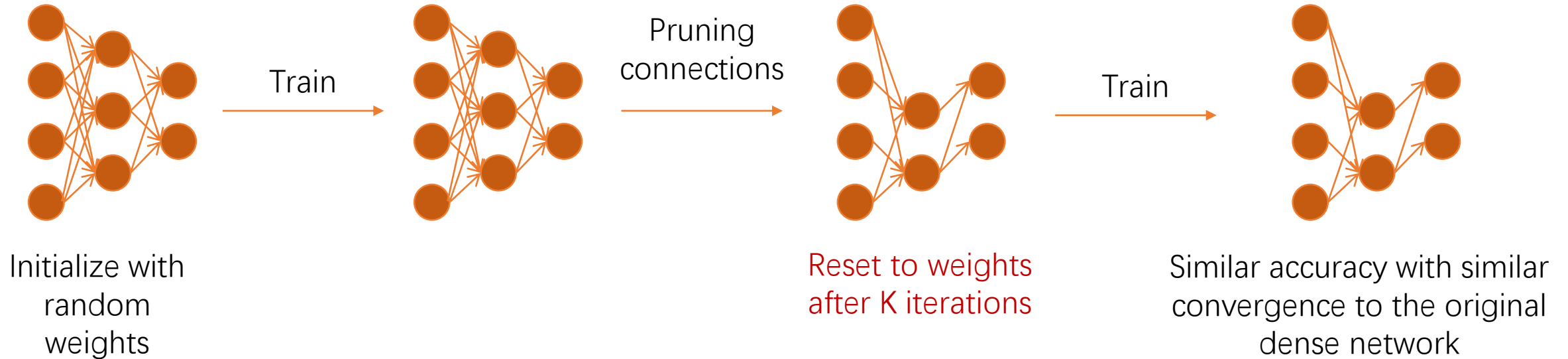
Lottery Ticket Hypothesis

- Winning ticket: a “lucky” sub-network that trains as fast as the original dense network;



Lottery Ticket Hypothesis

- Winning ticket: a “lucky” sub-network that trains as fast as the original dense network;



Frankle & Carbin (2019) — The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks.

Frankle et al. (2019) — Stabilizing the Lottery Ticket Hypothesis

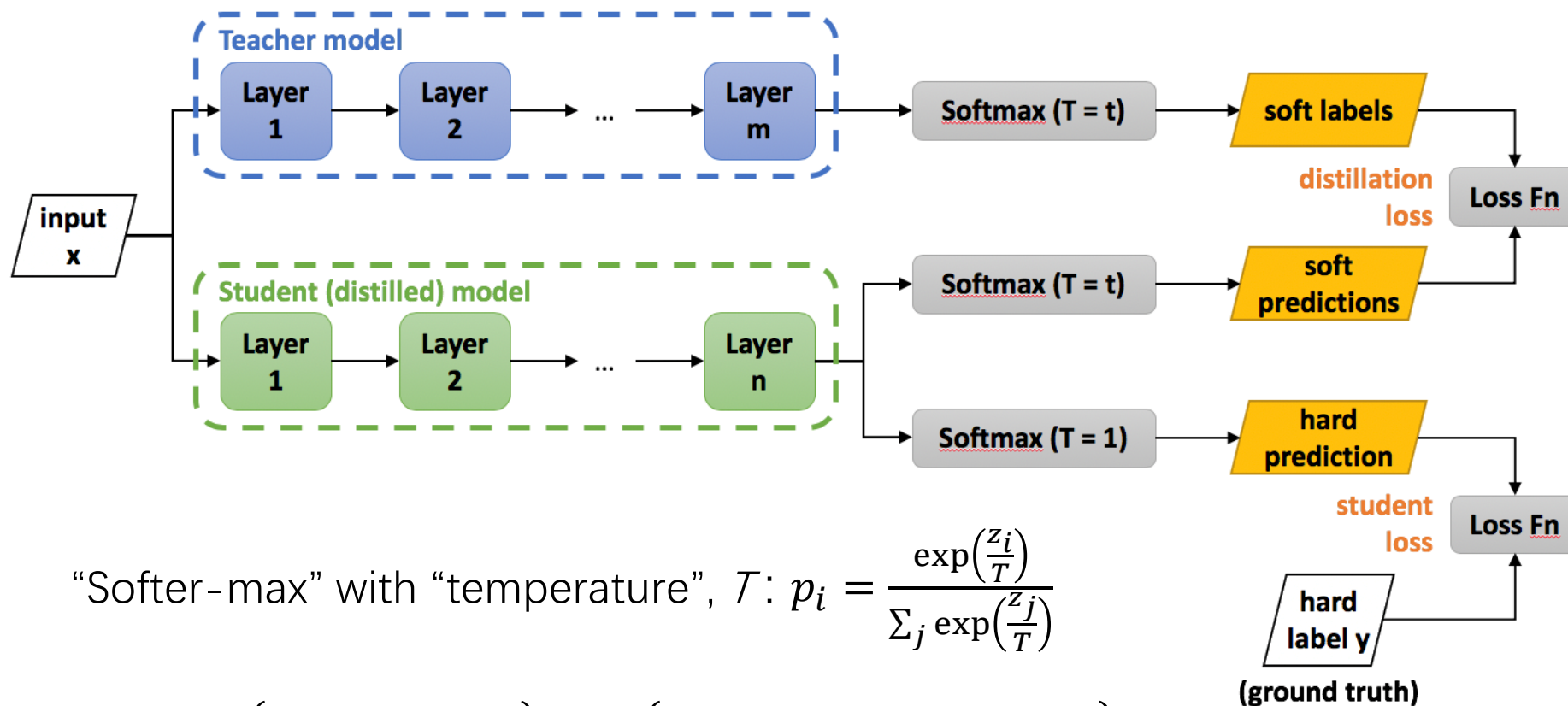
Zhou et al. (2019) — Deconstructing lottery tickets: Zeros, signs, and the supermask

Renda, Frankle & Carbin (2020) — "Comparing Rewinding and Fine-tuning in Neural Network Pruning"

立志成才 报国裕民

Other Compression Methods: Knowledge Distillation

- “Knowledge is transferred from the teacher model to the student model by minimizing a loss function.”



Other Methods: Low-Rank Approximation (LRA)

- The weight matrix in neural networks are highly redundant, thus leading to low rank;
- Compress the matrix with singular value decomposition (SVD)

$$W = USV^H$$

- W : $m * n$ matrix; U : $m * m$ matrix; S : diagonal matrix, $m * n$; V^H : $n * n$ matrix
- Low-rank approximation: truncate to r singular values/vectors;

- Example $W = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 4 & 3 & 2 \\ 1 & 3 & 5 & 7 \\ 4 & 3 & 2 & 1 \end{bmatrix}$

`torch.linalg.svd`

$$U = \begin{bmatrix} -0.41 & 0.26 & 0.56 & -0.67 \\ -0.51 & -0.58 & -0.53 & -0.35 \\ -0.67 & 0.57 & -0.25 & 0.41 \\ -0.36 & -0.53 & 0.58 & 0.51 \end{bmatrix}$$

$$\text{diag}(S) = [12.89 \quad 5.649 \quad 6.7 \times 10^{-7} \quad 5.4 \times 10^{-8}]$$

$$W' = \begin{bmatrix} 0.97 & 1.99 & 3.01 & 4.04 \\ 4.99 & 4.01 & 3.03 & 2.01 \\ 0.99 & 3.01 & 5.03 & 7.08 \\ 4.03 & 3.03 & 2.04 & 1.02 \end{bmatrix}$$

$$W = USV^H$$

$$V^H = \begin{bmatrix} -0.39 & -0.46 & -0.53 & -0.60 \\ -0.74 & -0.30 & 0.14 & 0.59 \\ 0.55 & 0.74 & 0.16 & 0.35 \\ -0.02 & 0.39 & 0.82 & 0.42 \end{bmatrix}$$

Low-Rank Approximation (LRA)

• Example $W = \begin{bmatrix} 0 & 0.1 & -0.1 \\ 1 & 1 & 1 \\ 0 & -0.1 & 0.1 \end{bmatrix}$

`torch.linalg.svd`

$$U_r^T = [0.0 \quad -1 \quad 3.75 \times 10^{-9}]$$

$$\text{diag}(S_r) = [1.732]$$

$$V_r^H = [-0.577 \quad -0.577 \quad -0.577]$$

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 4 & 3 & 2 \\ 1 & 3 & 5 & 7 \\ 4 & 3 & 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 12.1 & 9.1 \\ 9 & 15 \end{bmatrix} = X * W \approx U_r S_r V_r^H X = (X \otimes U_r) \otimes (V_r^H S_r) = \begin{bmatrix} 11.98 & 8.98 \\ 8.98 & 14.97 \end{bmatrix}$$



LoRA

- For fine-tuning of an NN model.

$$W = W_{pre-train} + \frac{\alpha}{r \text{ or } \sqrt{r}} AB$$

- A B are the fine-tuning matrix after decomposition, for example

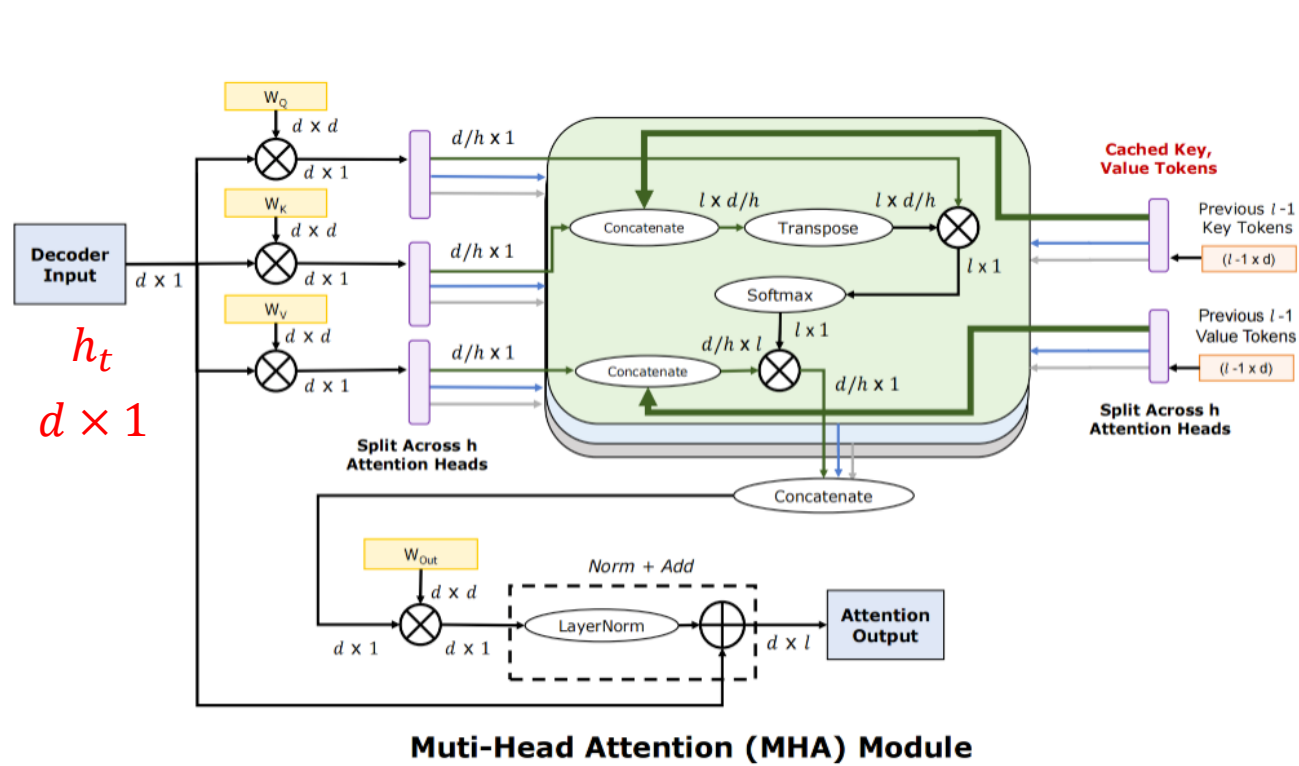
$$W = \begin{matrix} & \begin{matrix} w_{11} & w_{12} & w_{13} \end{matrix} \\ \begin{matrix} w_{21} \\ w_{31} \end{matrix} & \begin{matrix} w_{22} & w_{23} \\ w_{32} & w_{33} \end{matrix} \end{matrix} \quad A = \begin{matrix} a_1 \\ a_2 \\ a_3 \end{matrix} \quad B = \begin{matrix} b_1 \\ b_2 \\ b_3 \end{matrix}$$

- Fix $W_{pre-train}$ and train A B instead. For the current layer

$$Y = WX = W_{pre-train}X + \frac{\alpha}{r \text{ or } \sqrt{r}} XAB$$



DeepSeek v2 - MLA



$$h_t$$

$$d \times 1$$

$$W^{DKV}$$

$$d \times d_c$$

$$W^{DQ}$$

$$d \times d'_c$$

Compress
matrices

$$c_t^{KV}$$

$$1 \times d_c$$

$$c_t^Q$$

$$1 \times d'_c$$

$$W^{UK}$$

$$n_h d_h \times d_c$$

$$W^{UV}$$

$$n_h d_h \times d_c$$

$$W^{UQ}$$

$$d_q \times d'_c$$

Decompress
matrices

$$W^K \approx W^{DKV} W^{UK} \quad W^V \approx W^{DKV} W^{UV} \quad W^Q \approx W^{DQ} W^{UQ}$$

Low-rank Approximation (Cont'd)

- Trace back to Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation (Denton, Zaremba, Bruna, LeCun, Fergus, NIPS 2014)
 - SVD or outer product decomposition
 - Similar works apply truncated SVD to address FC layers during 2013-2014
- Other decomposition methods emerge such as Tucker/CP decomposition;
- Adopted in transformers such as LoRA fine-tuning (2021), Linformer (2020)
- Can learn rank per layer (adaptively)



Summary

- Model compression can be achieved by pruning, distillation, LRA and quantization;
- They can be effective in reducing hardware/energy cost, improving performance
- Various strategies can be used;
- An active field in ML research
 - How to deal with the outliers for quantization; Reduce overheads for quantization;
 - How to prune a NN model (when, what, how, how often);
 - Sparsity hardware (will be covered later)

