# CENG 5030
# Energy Efficient Computing

## Mo03: Quantization

Bei Yu
CSE Department, CUHK
`byu@cse.cuhk.edu.hk`

(Latest update: October 30, 2023)

2023 Fall

## These slides contain/adapt materials developed by

- Hardware for Machine Learning, Shao Spring 2020 @ UCB
- 8-bit Inference with TensorRT
- Amir Gholami et al. (2021). “A survey of quantization methods for efficient neural network inference”. In: *arXiv preprint*

# Overview

1. Floating Point Number
2. Integer & Fixed-Point Number
3. Quantization Overview
4. Quantization – First Example
5. Post Training Quantization (PTQ)
6. Quantization Aware Training (QAT)

# Floating Point Number

# Floating Point Number

Scientific notation: $6.6254 \times 10^{-27}$

- A normalized number of certain accuracy (e.g. 6.6254 is called the mantissa)
- Scale factors to determine the position of the decimal point (e.g. $10^{-27}$ indicates position of decimal point and is called the exponent; the **base** is implied)
- Sign bit

# Normalized Form

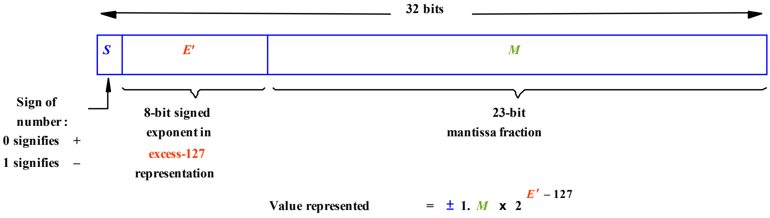
- Floating Point Numbers can have multiple forms, e.g.

$$\begin{aligned} 0.232 \times 10^4 &= 2.32 \times 10^3 \\ &= 23.2 \times 10^2 \\ &= 2320. \times 10^0 \\ &= 232000. \times 10^{-2} \end{aligned}$$

- It is desirable for each number to have a unique representation => Normalized Form
- We normalize Mantissa's in the Range $[1..R)$, where R is the Base, e.g.:
  - [1..2) for BINARY
  - [1..10) for DECIMAL

# IEEE Standard 754 Single Precision

32-bit, float in C / C++ / Java



32 bits

S | E′ | M

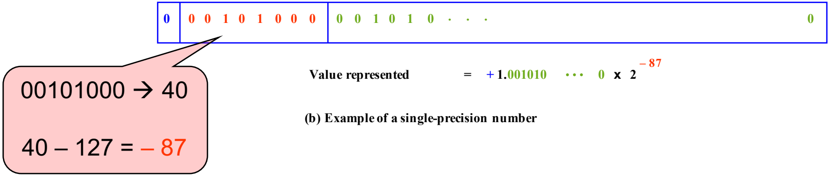
Sign of number:
0 signifies +
1 signifies –

8-bit signed exponent in excess-127 representation

23-bit mantissa fraction

Value represented = $\pm 1.M \times 2^{E' - 127}$

(a) Single precision



0 | 0 0 1 0 1 0 0 0 | 0 0 1 0 1 0 · · · 0

Value represented = $+1.001010 \cdots 0 \times 2^{-87}$

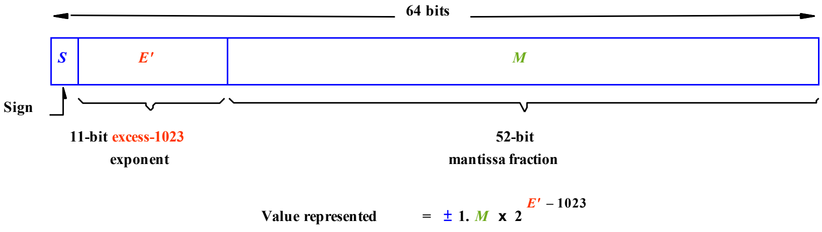
00101000 → 40

40 – 127 = – 87

(b) Example of a single-precision number

# IEEE Standard 754 Double Precision

64-bit, float in C / C++ / Java



Value represented $= \pm 1.M \times 2^{E' - 1023}$

(c) Double precision

**Question:**

What is the IEEE single precision number $40C0\ 0000_{16}$ in decimal?

**Question**:

What is -$0.5_{10}$ in IEEE single precision binary floating point format?

# Special Values

Exponents of all 0's and all 1's have special meaning

- E=0, M=0 represents 0 (sign bit still used so there is ±0)
- E=0, M≠0 is a denormalized number $\pm 0.M \times 2^{-126}$ (smaller than the smallest normalized number)
- E=All 1's, M=0 represents ±`Infinity`, depending on Sign
- E=All 1's, M≠0 represents `NaN`

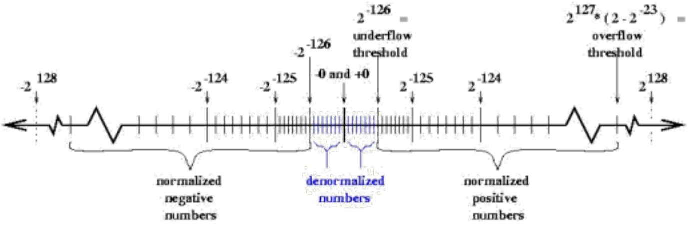
# Ref: IEEE Standard 754 Numbers

- Normalized +/– 1.d…d x $2^{exp}$
- ***Denormalized*** +/– 0.d…d x $2^{min\_exp}$ → to represent *near-zero* numbers
  e.g. + 0.0000…0000001 x $2^{-126}$ for Single Precision

| **Format** | **# bits** | **# significant bits** | **macheps** | **# exponent bits** | **exponent range** |
|---|---|---|---|---|---|
| **Single** | **32** | **1+23** | $2^{-24}$ ($\sim10^{-7}$) | **8** | $2^{-126} - 2^{+127}$ ($\sim10^{\pm38}$) |
| **Double** | **64** | **1+52** | $2^{-53}$ ($\sim10^{-16}$) | **11** | $2^{-1022} - 2^{+1023}$ ($\sim10^{\pm308}$) |
| **Double Extended** | **>=80** | **>=64** | $<=2^{-64}$($\sim10^{-19}$) | **>=15** | $2^{-16382} - 2^{+16383}$ ($\sim10^{\pm4932}$) |

**(Double Extended is *80 bits on all Intel machines*)**

**macheps = Machine Epsilon =** $\varepsilon_{mach}$ **=** $2^{-(\text{\# significand bits})}$



# Inaccurate Floating Point Operations

## Example: Find 1st root of a quadratic equation[1]

$$r = \frac{-b + \sqrt{b^2 - 4 \cdot a \cdot c)}}{2 \cdot a}$$

**Expected:** 0.00023025562642476431

**Double:** 0.00023025562638524986

**Float:** 0.00024670246057212353

---

[1] On Sparc processor, Solaris, gcc 3.3 (ANSI C)

# Inaccurate Floating Point Operations

## Example: Find 1st root of a quadratic equation[1]

$$r = \frac{-b + \sqrt{b^2 - 4 \cdot a \cdot c)}}{2 \cdot a}$$

**Expected:** 0.00023025562642476431

**Double:** 0.00023025562638524986

**Float:** 0.00024670246057212353

- Problem is that if c is near zero, $\sqrt{b^2 - 4 \cdot a \cdot c} \approx b$
- Rule of thumb: use the highest precision which does not give up too much speed

---

[1] On Sparc processor, Solaris, gcc 3.3 (ANSI C)

# Integer & Fixed-Point Number

# Unsigned Binary Representation

| Hex | Binary | Decimal |
|---|---|---|
| 0x00000000 | 0…0000 | 0 |
| 0x00000001 | 0…0001 | 1 |
| 0x00000002 | 0…0010 | 2 |
| 0x00000003 | 0…0011 | 3 |
| 0x00000004 | 0…0100 | 4 |
| 0x00000005 | 0…0101 | 5 |
| 0x00000006 | 0…0110 | 6 |
| 0x00000007 | 0…0111 | 7 |
| 0x00000008 | 0…1000 | 8 |
| 0x00000009 | 0…1001 | 9 |
|  | … |  |
| 0xFFFFFFFC | 1…1100 | $2^{32} - 4$ |
| 0xFFFFFFFD | 1…1101 | $2^{32} - 3$ |
| 0xFFFFFFFE | 1…1110 | $2^{32} - 2$ |
| 0xFFFFFFFF | 1…1111 | $2^{32} - 1$ |

|  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
| $2^{31}$ | $2^{30}$ | $2^{29}$ | . . . | $2^{3}$ | $2^{2}$ | $2^{1}$ | $2^{0}$ | bit weight |
| 31 | 30 | 29 | . . . | 3 | 2 | 1 | 0 | bit position |
| **1** | **1** | **1** | **. . .** | **1** | **1** | **1** | **1** | **bit** |

⇩

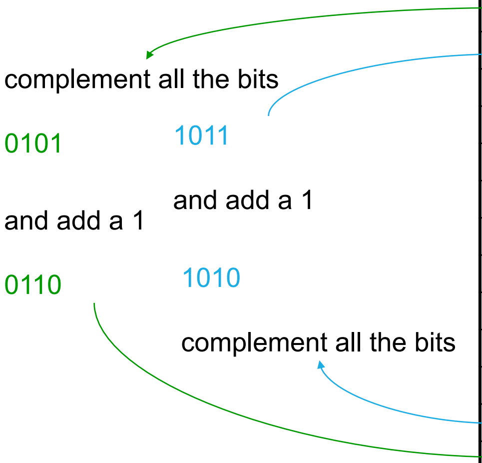
**1 0 0 0 . . . 0 0 0 0 - 1**

⇩

$\mathbf{2^{32} - 1}$

# Signed Binary Representation

|  | 2’sc binary | decimal |
|---|---|---|
| $-2^3$ = | 1000 | -8 |
| $-(2^3 - 1)$ = | 1001 | -7 |
|  | 1010 | -6 |
|  | 1011 | -5 |
|  | 1100 | -4 |
|  | 1101 | -3 |
|  | 1110 | -2 |
|  | 1111 | -1 |
|  | 0000 | 0 |
|  | 0001 | 1 |
|  | 0010 | 2 |
|  | 0011 | 3 |
|  | 0100 | 4 |
|  | 0101 | 5 |
|  | 0110 | 6 |
| $2^3 - 1$ = | 0111 | 7 |



complement all the bits

0101

and add a 1

0110

1011

and add a 1

1010

complement all the bits

# Fixed-Point Arithmetic

- Integers with a binary point and a bias
  - “slope and bias”: y = s*x + z
  - Qm.n: m (# of integer bits) n (# of fractional bits)

s = 1, z = 0

| 2^2 | 2^1 | 2^0 | Val |
|---|---|---|---|
| 0 | 0 | 0 | **0** |
| 0 | 0 | 1 | **1** |
| 0 | 1 | 0 | **2** |
| 0 | 1 | 1 | **3** |
| 1 | 0 | 0 | **4** |
| 1 | 0 | 1 | **5** |
| 1 | 1 | 0 | **6** |
| 1 | 1 | 1 | **7** |

s = 1/4, z = 0

| 2^0 | 2^-1 | 2^-2 | Val |
|---|---|---|---|
| 0 | 0 | 0 | **0** |
| 0 | 0 | 1 | **1/4** |
| 0 | 1 | 0 | **2/4** |
| 0 | 1 | 1 | **3/4** |
| 1 | 0 | 0 | **1** |
| 1 | 0 | 1 | **5/4** |
| 1 | 1 | 0 | **6/4** |
| 1 | 1 | 1 | **7/4** |

s = 4, z = 0

| 2^4 | 2^3 | 2^2 | Val |
|---|---|---|---|
| 0 | 0 | 0 | **0** |
| 0 | 0 | 1 | **4** |
| 0 | 1 | 0 | **8** |
| 0 | 1 | 1 | **12** |
| 1 | 0 | 0 | **16** |
| 1 | 0 | 1 | **20** |
| 1 | 1 | 0 | **24** |
| 1 | 1 | 1 | **28** |

s = 1.5, z =10

| 2^2 | 2^1 | 2^0 | Val |
|---|---|---|---|
| 0 | 0 | 0 | **1.5*0 +10** |
| 0 | 0 | 1 | **1.5*1 +10** |
| 0 | 1 | 0 | **1.5*2 +10** |
| 0 | 1 | 1 | **1.5*3 +10** |
| 1 | 0 | 0 | **1.5*4 +10** |
| 1 | 0 | 1 | **1.5*5 +10** |
| 1 | 1 | 0 | **1.5*6 +10** |
| 1 | 1 | 1 | **1.5*7 +10** |

# Catastrophic Cancellation

$(a - b)$ is inaccurate when $a >> b$ or $a << b$

## Decimal Example 1:

- Using 2 significant digits
- Compute mean of 5.1 and 5.2 using the formula $(a + b)/2$:
- $a + b = 10$ (with 2 significant digits, 10.3 can only be stored as 10)
- $10/2 = 5.0$ (the computed mean is less than both numbers!!!)

## Decimal Example 2:

- Using 8 significant digits to compute sum of three numbers:
- $(11111113 + (-11111111)) + 7.5111111 = 9.5111111$
- $11111113 + ((-11111111) + 7.5111111) = 10.000000$

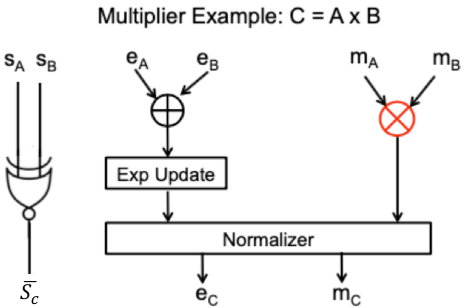
# Catastrophic Cancellation

## Catastrophic cancellation occurs when

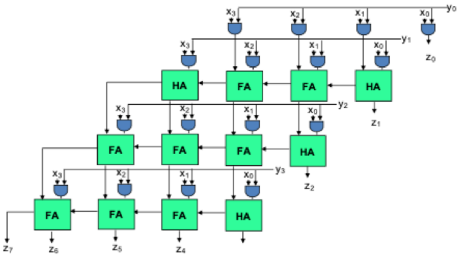
$$\left| \frac{[\text{round}(x) \times \text{round}(y)] - \text{round}(x \times y)}{\text{round}(x \times y)} \right| >> \epsilon$$

# Hardware Implications

## Multipliers

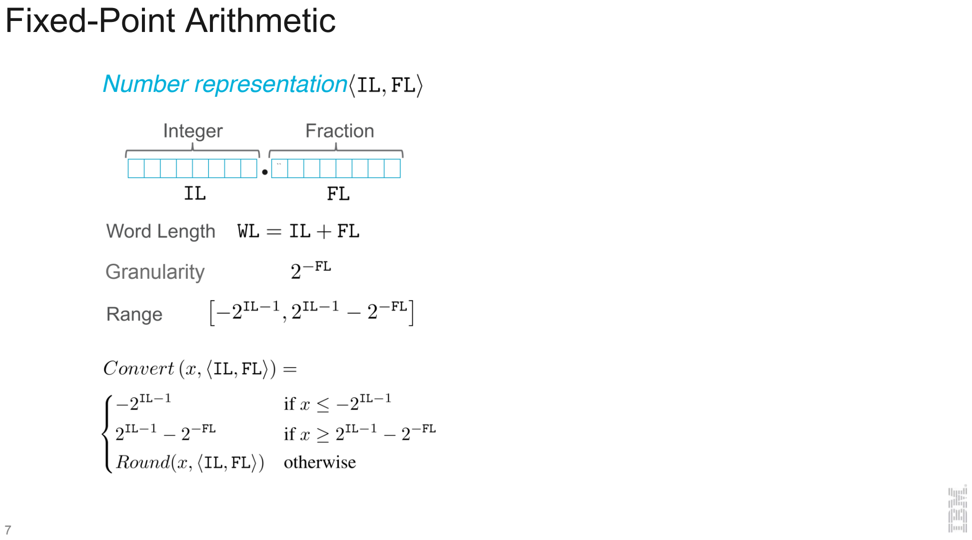


Floating-point multiplier



Fixed-point multiplier

# Case Study: ICML 2015[2]



## Fixed-Point Arithmetic

*Number representation* $\langle \texttt{IL}, \texttt{FL} \rangle$

Word Length $\texttt{WL} = \texttt{IL} + \texttt{FL}$

Granularity $2^{-\texttt{FL}}$

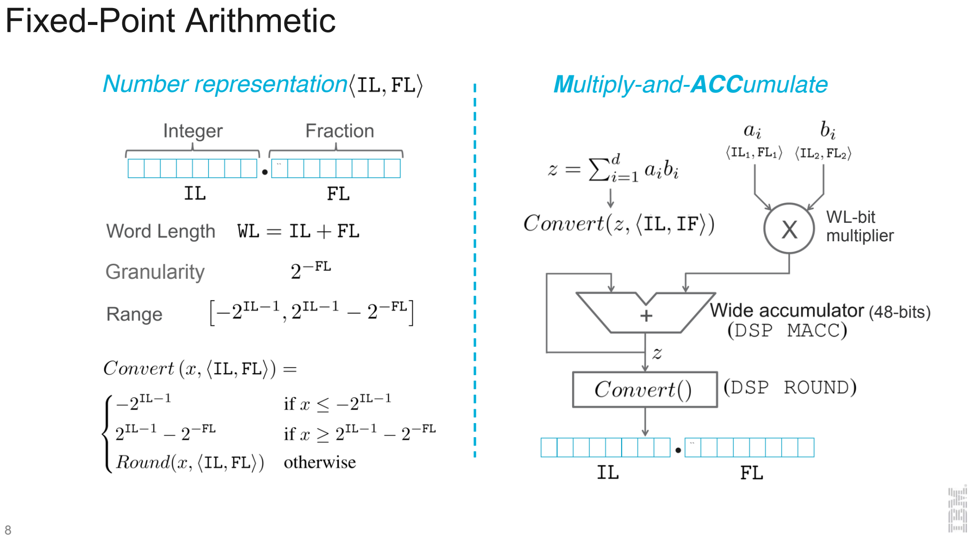
Range $\left[-2^{\texttt{IL}-1}, 2^{\texttt{IL}-1} - 2^{-\texttt{FL}}\right]$

$$Convert\left(x, \langle \texttt{IL}, \texttt{FL} \rangle\right) = \begin{cases} -2^{\texttt{IL}-1} & \text{if } x \le -2^{\texttt{IL}-1} \\ 2^{\texttt{IL}-1} - 2^{-\texttt{FL}} & \text{if } x \ge 2^{\texttt{IL}-1} - 2^{-\texttt{FL}} \\ Round(x, \langle \texttt{IL}, \texttt{FL} \rangle) & \text{otherwise} \end{cases}$$

---

[2] Suyog Gupta et al. (2015). "Deep learning with limited numerical precision". In: *Proc. ICML*, pp. 1737–1746.

# Case Study: ICML 2015[2]



## Fixed-Point Arithmetic

*Number representation* $\langle \mathtt{IL}, \mathtt{FL} \rangle$

Word Length $\mathtt{WL} = \mathtt{IL} + \mathtt{FL}$

Granularity $2^{-\mathtt{FL}}$

Range $\left[-2^{\mathtt{IL}-1}, 2^{\mathtt{IL}-1} - 2^{-\mathtt{FL}}\right]$

$$Convert\left(x, \langle \mathtt{IL}, \mathtt{FL} \rangle\right) = \begin{cases} -2^{\mathtt{IL}-1} & \text{if } x \le -2^{\mathtt{IL}-1} \\ 2^{\mathtt{IL}-1} - 2^{-\mathtt{FL}} & \text{if } x \ge 2^{\mathtt{IL}-1} - 2^{-\mathtt{FL}} \\ Round(x, \langle \mathtt{IL}, \mathtt{FL} \rangle) & \text{otherwise} \end{cases}$$

*Multiply-and-**ACC**umulate*

$$z = \sum_{i=1}^{d} a_i b_i$$

$$\downarrow$$

$$Convert(z, \langle \mathtt{IL}, \mathtt{IF} \rangle)$$

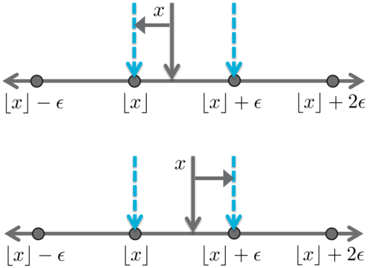
---

[2] Suyog Gupta et al. (2015). “Deep learning with limited numerical precision”. In: *Proc. ICML*, pp. 1737–1746.

# Case Study: ICML 2015[2]

## Fixed-Point Arithmetic: Rounding Modes

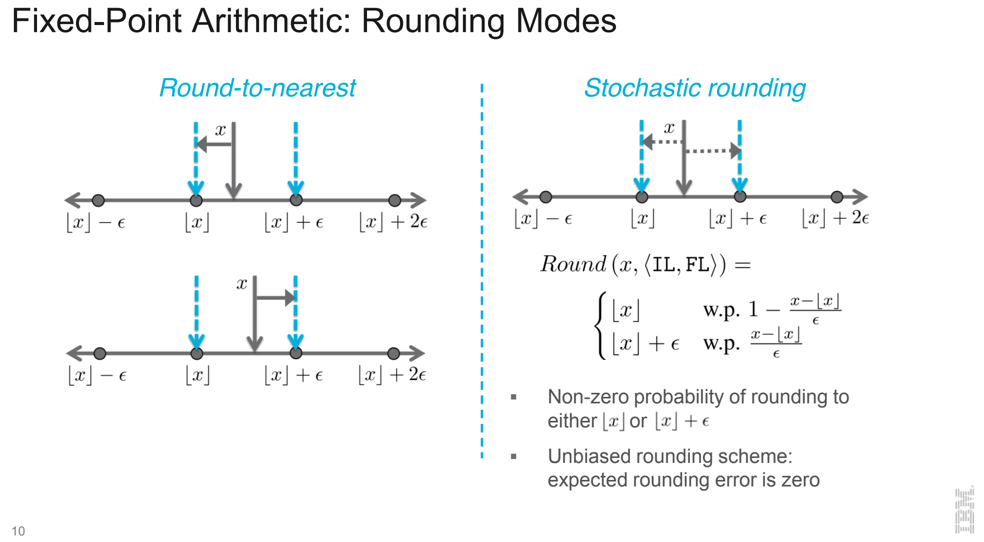
*Round-to-nearest*



---

[2] Suyog Gupta et al. (2015). "Deep learning with limited numerical precision". In: *Proc. ICML*, pp. 1737–1746.

# Case Study: ICML 2015[2]



## Fixed-Point Arithmetic: Rounding Modes

*Round-to-nearest*

*Stochastic rounding*

$$Round\left(x, \langle \mathtt{IL}, \mathtt{FL} \rangle\right) = \begin{cases} \lfloor x \rfloor & \text{w.p. } 1 - \frac{x - \lfloor x \rfloor}{\epsilon} \\ \lfloor x \rfloor + \epsilon & \text{w.p. } \frac{x - \lfloor x \rfloor}{\epsilon} \end{cases}$$

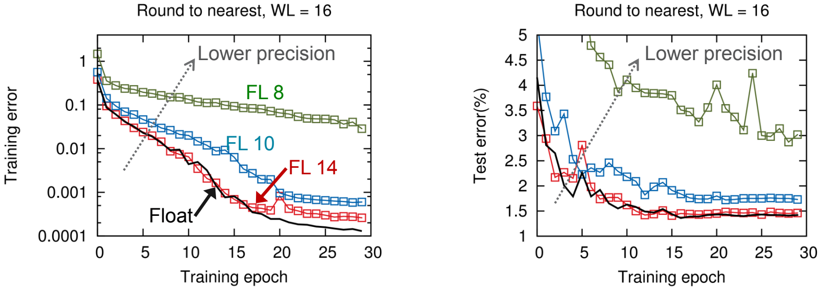
- Non-zero probability of rounding to either $\lfloor x \rfloor$ or $\lfloor x \rfloor + \epsilon$
- Unbiased rounding scheme: expected rounding error is zero

---

[2] Suyog Gupta et al. (2015). “Deep learning with limited numerical precision”. In: *Proc. ICML*, pp. 1737–1746.

# Case Study: ICML 2015[2]

## MNIST: *Fully-connected DNNs*

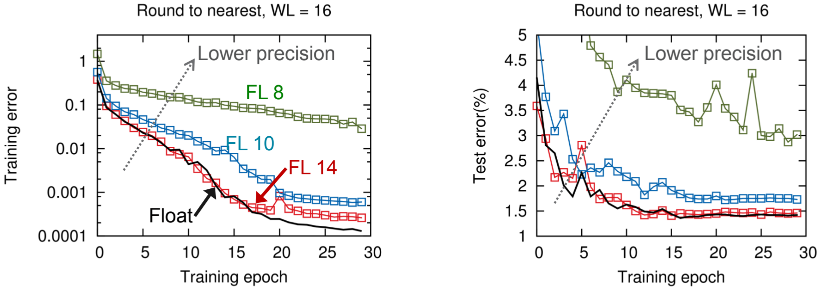


---

[2] Suyog Gupta et al. (2015). “Deep learning with limited numerical precision”. In: *Proc. ICML*, pp. 1737–1746.

# Case Study: ICML 2015[2]

## MNIST: *Fully-connected DNNs*

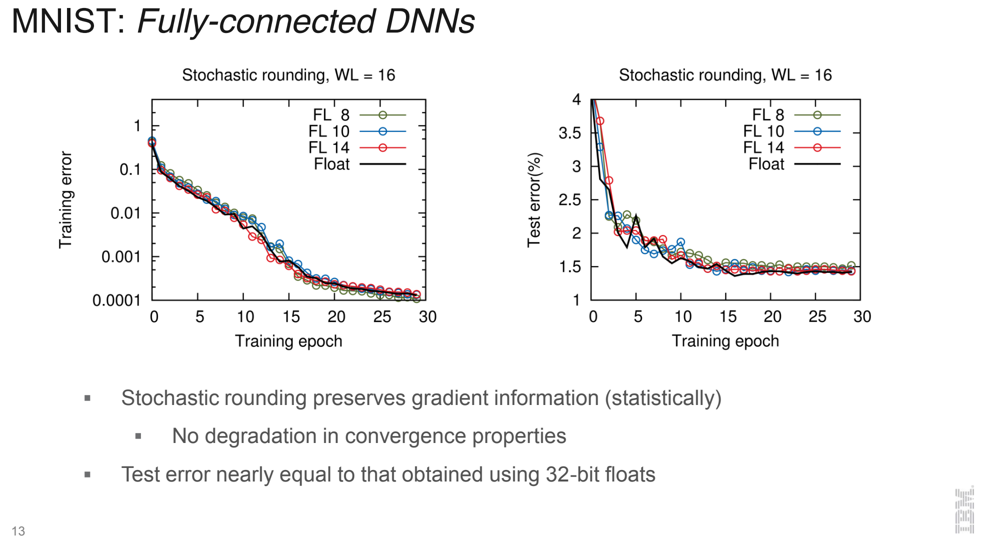


- For small fractional lengths (FL < 12), a large majority of weight updates are rounded to zero when using the round-to-nearest scheme.
  - Convergence slows down
- For FL < 12, there is a noticeable degradation in the classification accuracy

---

[2] Suyog Gupta et al. (2015). “Deep learning with limited numerical precision”. In: *Proc. ICML*, pp. 1737–1746.

# Case Study: ICML 2015[2]



## MNIST: *Fully-connected DNNs*

- Stochastic rounding preserves gradient information (statistically)
  - No degradation in convergence properties
- Test error nearly equal to that obtained using 32-bit floats

---

[2] Suyog Gupta et al. (2015). “Deep learning with limited numerical precision”. In: *Proc. ICML*, pp. 1737–1746.

# Quantization Overview

# Quantization in DNN

**Quantization:**

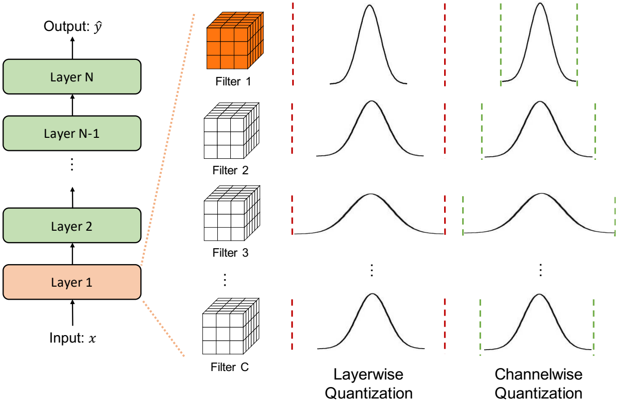
$$Q(r) = \text{Int}(r/S) - Z$$

**Dequantization:**

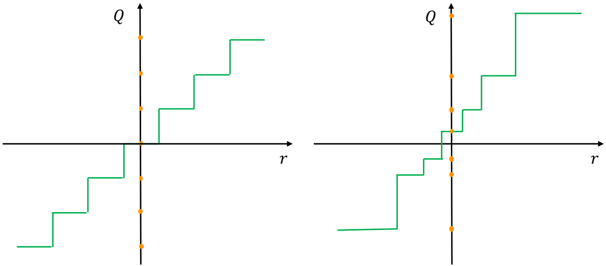
$$\hat{r} = S(Q(r) + Z)$$

Granularity:

- Layerwise
- Groupwise
- Channelwise



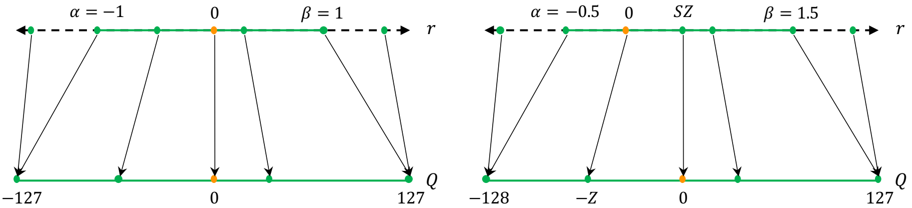
# Uniform vs. Non-Uniform



(a) uniform quantization (b) uniform quantization

- Real values in the continuous domain $r$ are mapped into discrete
- Lower precision values in the quantized domain $Q$.
- **Uniform** quantization: distances between quantized values are the same
- **Non-uniform** quantization: distances between quantized values can vary

# Symmetric vs. Asymmetric

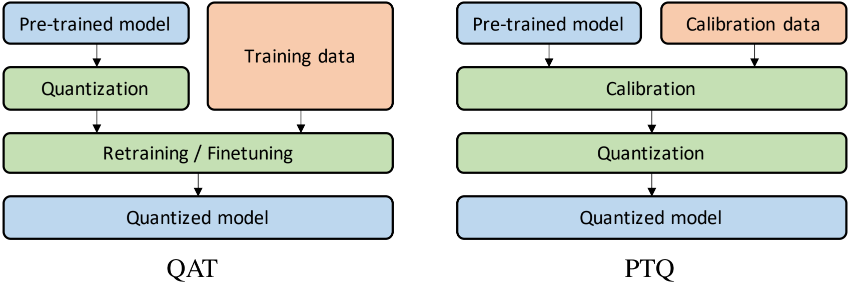


(a) Symmetric quantization

(b) Asymmetric quantization

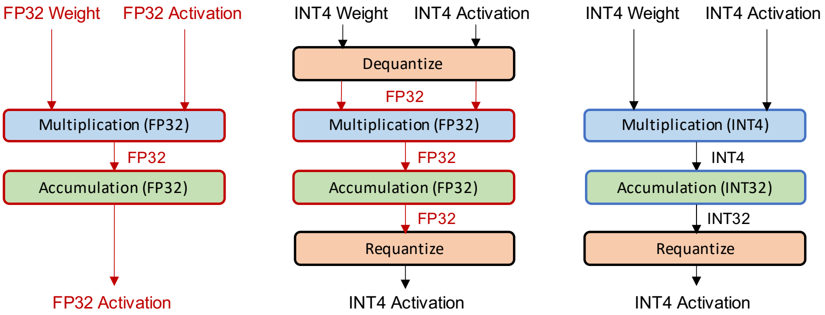
- Symmetric vs. Asymmetric: $Z = 0$ ?
- Fig. (a) Symmetric w. restricted range maps [-127, 127],
- Fig. (b) Asymmetric w. full range maps to [-128, 127]
- Both for 8-bit quantization case.

# QAT and PTQ



- quantization-aware training (QAT): model is quantized using training data to adjust parameters and recover accuracy degradation.
- post-training quantization (PTQ): a pre-trained model is calibrated using finetuning data (e.g., a small subset of training data) to compute the clipping ranges and the scaling factors.
- Key difference: Model parameters fixed/unfixed.

# Simulated quantization vs Integer-Only quantization



**Left** : Full-precision

**Middle** : Simulated quantization

**Right** : Integer-only quantization

# Backend Support for Quantization Deployment

## Hardware Support

- Nvidia GPU: Tensor Core support FP16, Int8 and Int4
- Arm: Neon 128-bit SIMD instruction: 4×32bit or 8×16bit up to 16×8bit
- Intel: SSE intrinsics, same as above
- DSP, AI Chip

## Some common architectures:

- For CPU: Tensorflow Lite, QNNPACK, NCNN
- For GPU: TensorRT
- Versatile Compiler such TVM.qnn

# Quantization – First Example

# Linear quantization

**Representation:**

Tensor Values = FP32 scale factor * int8 array + FP32 bias

# Do we really need bias?

Two matrices:

```
A = scale_A * QA + bias_A
B = scale_B * QB + bias_B
```

Let’s multiply those 2 matrices:

```
A * B = scale_A * scale_B * QA * QB +
        scale_A * QA * bias_B       +
        scale_B * QB * bias_A       +
        bias_A  * bias_B
```

# Do we really need bias?

**Two matrices:**

A = scale_A * QA + ~~bias_A~~

B = scale_B * QB + ~~bias_B~~

**Let’s multiply those 2 matrices:**

A * B = scale_A * scale_B * QA * QB +

~~scale_A * QA * bias_B~~ ~~+~~

~~scale_B * QB * bias_A~~ ~~+~~

~~bias_A * bias_B~~

# Do we really need bias? No!

Two matrices:

```
A = scale_A * QA
B = scale_B * QB
```

Let’s multiply those 2 matrices:

```
A * B = scale_A * scale_B * QA * QB
```

# Symmetric linear quantization

Representation:

Tensor Values = FP32 scale factor * int8 array

One FP32 scale factor for the entire int8 tensor

**Q: How do we set scale factor?**

# MINIMUM QUANTIZED VALUE

- Integer range is not completely symmetric. E.g. in 8bit, [-128, 127]
  - If use [-127, 127], $s = \frac{127}{\alpha}$
    - Range is symmetric
    - 1/256 of int8 range is not used. 1/16 of int4 range is not used
  - If use full range [-128, 127], $s = \frac{128}{\alpha}$
    - Values should be quantized to 128 will be clipped to 127
    - Asymmetric range may introduce bias

# EXAMPLE OF QUANTIZATION BIAS

Bias introduced when int values are in [-128, 127]

$$A = [-2.2 \quad -1.1 \quad 1.1 \quad 2.2]\ ,\ B = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.3 \\ 0.5 \end{bmatrix},\ AB = 0$$

8bit scale quantization, use [-128, 127]. $s_A$=128/2.2, $s_B$=128/0.5

$$[-128 \quad -64 \quad 64 \quad 127] * \begin{bmatrix} 127 \\ 77 \\ 77 \\ 127 \end{bmatrix} = -127$$

Dequantize -127 will get -0.00853. A small bias is introduced towards -∞

# EXAMPLE OF QUANTIZATION BIAS

## No bias when int values are in [-127, 127]

$$A = \begin{bmatrix} -2.2 & -1.1 & 1.1 & 2.2 \end{bmatrix},\ B = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.3 \\ 0.5 \end{bmatrix},\ AB = 0$$

8-bit scale quantization, use [-127, 127]. $s_A$=127/2.2, $s_B$=127/0.5

$$\begin{bmatrix} -127 & -64 & 64 & 127 \end{bmatrix} * \begin{bmatrix} 127 \\ 76 \\ 76 \\ 127 \end{bmatrix} = 0$$

Dequantize 0 will get 0

# MATRIX MULTIPLY EXAMPLE

## Scale Quantization

$$\begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 0.65 \end{pmatrix} * \begin{pmatrix} 0.35 \\ -0.51 \end{pmatrix} = \begin{pmatrix} -0.651 \\ -0.423 \end{pmatrix}$$

# MATRIX MULTIPLY EXAMPLE

## Scale Quantization

$$\begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 0.65 \end{pmatrix} * \begin{pmatrix} 0.35 \\ -0.51 \end{pmatrix} = \begin{pmatrix} -0.651 \\ -0.423 \end{pmatrix}$$

8bit quantization

choose [-2, 2] fp range (scale 127/2=63.5) for first matrix and [-1, 1] fp range (scale = 127/1=127) for the second

$$\begin{pmatrix} -98 & 14 \\ -17 & 41 \end{pmatrix} * \begin{pmatrix} 44 \\ -65 \end{pmatrix} = \begin{pmatrix} -5222 \\ -3413 \end{pmatrix}$$

# MATRIX MULTIPLY EXAMPLE

## Scale Quantization

$$\begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 0.65 \end{pmatrix} * \begin{pmatrix} 0.35 \\ -0.51 \end{pmatrix} = \begin{pmatrix} -0.651 \\ -0.423 \end{pmatrix}$$

8bit quantization

choose [-2, 2] fp range (scale 127/2=63.5) for first matrix and [-1, 1] fp range (scale = 127/1=127) for the second

$$\begin{pmatrix} -98 & 14 \\ -17 & 41 \end{pmatrix} * \begin{pmatrix} 44 \\ -65 \end{pmatrix} = \begin{pmatrix} -5222 \\ -3413 \end{pmatrix}$$

The result has an overall scale of 63.5*127 . We can *dequantize* back to float

$$\begin{pmatrix} -5222 \\ -3413 \end{pmatrix} * \frac{1}{63.5 * 127} = \begin{pmatrix} -0.648 \\ -0.423 \end{pmatrix}$$

# REQUANTIZE

## Scale Quantization

$$\begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 0.65 \end{pmatrix} * \begin{pmatrix} 0.35 \\ -0.51 \end{pmatrix} = \begin{pmatrix} -0.651 \\ -0.423 \end{pmatrix}$$

8bit quantization

choose [-2, 2] fp range for first matrix and [-1, 1] fp range for the second

$$\begin{pmatrix} -98 & 14 \\ -17 & 41 \end{pmatrix} * \begin{pmatrix} 44 \\ -65 \end{pmatrix} = \begin{pmatrix} -5222 \\ -3413 \end{pmatrix}$$

*Requantize* output to a different quantized representation with fp range [-3, 3]:

$$\begin{pmatrix} -5222 \\ -3413 \end{pmatrix} * \frac{127/3}{63.5 * 127} = \begin{pmatrix} -27 \\ -18 \end{pmatrix}$$

# Post Training Quantization (PTQ)

# Greedy Layer-wise Quantization[3]

- For a fixed-point number, it representation is:

$$n = \sum_{i=0}^{bw-1} B_i \cdot 2^{-f_l} \cdot 2^i,$$

  where $bw$ is the bit width and $f_l$ is the fractional length which is dynamic for different layers and feature map sets while static in one layer.

- Weight quantization: find the optimal $f_l$ for weights:

$$f_l = \arg\min_{f_l} \sum |W_{float} - W(bw, f_l)|,$$

  where $W$ is a weight and $W(bw, f_l)$ represents the fixed-point format of $W$ under the given $bw$ and $f_l$.

---

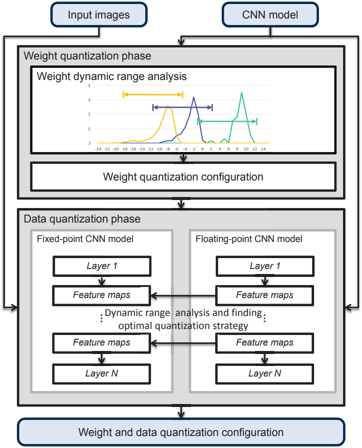
[3] Jiantao Qiu et al. (2016). "Going deeper with embedded fpga platform for convolutional neural network". In: *Proc. FPGA*, pp. 26–35.

# Greedy Layer-wise Quantization

- Feature quantization: find the optimal $f_l$ for features:

$$f_l = \arg\min_{f_l} \sum |x^+_{float} - x^+(bw, f_l)|,$$

where $x^+$ represents the result of a layer when we denote the computation of a layer as $x^+ = A \cdot x$.



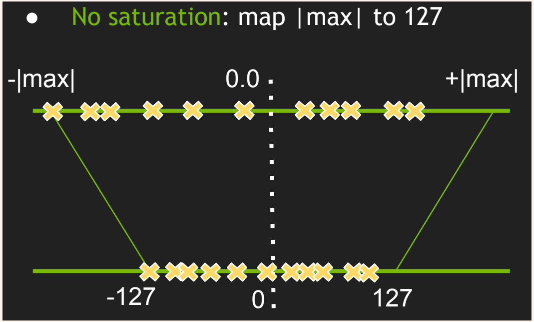
# Dynamic-Precision Data Quantization Results

| Network | VGG16 | | | | | | |
|---|---|---|---|---|---|---|---|
| Data Bits | Single-float | 16 | 16 | 8 | 8 | 8 | 8 |
| Weight Bits | Single-float | 16 | 8 | 8 | 8 | 8 | 8 or 4 |
| Data Precision | N/A | $2^{-2}$ | $2^{-2}$ | Impossible | $2^{-5}/2^{-1}$ | Dynamic | Dynamic |
| Weight Precision | N/A | $2^{-15}$ | $2^{-7}$ | Impossible | $2^{-7}$ | Dynamic | Dynamic |
| Top-1 Accuracy | 68.1% | 68.0% | 53.0% | Impossible | 28.2% | 66.6% | 67.0% |
| Top-5 Accuracy | 88.0% | 87.9% | 76.6% | Impossible | 49.7% | 87.4% | 87.6% |

| Network | CaffeNet | | | VGG16-SVD | | |
|---|---|---|---|---|---|---|
| Data Bits | Single-float | 16 | 8 | Single-float | 16 | 8 |
| Weight Bits | Single-float | 16 | 8 | Single-float | 16 | 8 or 4 |
| Data Precision | N/A | Dynamic | Dynamic | N/A | Dynamic | Dynamic |
| Weight Precision | N/A | Dynamic | Dynamic | N/A | Dynamic | Dynamic |
| Top-1 Accuracy | 53.9% | 53.9% | 53.0% | 68.0% | 64.6% | 64.1% |
| Top-5 Accuracy | 77.7% | 77.1% | 76.6% | 88.0% | 86.7% | 86.3% |

# Industrial Implementations – Nvidia TensorRT

## No Saturation Quantization – INT8 Inference

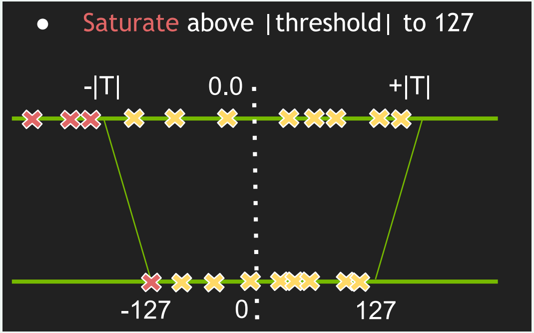


- No saturation: map |max| to 127

- Map the maximum value to 127, with unifrom step length.
- Suffer from outliers.

# Industrial Implementations – Nvidia TensorRT

## Saturation Quantization – INT8 Inference



- Saturate above |threshold| to 127

- Set a threshold as the maxiumum value.
- Divide the value domain into 2048 groups.
- Traverse all the possible thresholds to find the best one with minimum KL divergence.

# Industrial Implementations – Nvidia TensorRT

## Relative Entropy of two encodings

- INT8 model encodes the same information as the original FP32 model.
- Minimize the loss of information.
- Loss of information is measured by Kullback-Leibler divergence (*a.k.a.*, relative entropy or information divergence).
  - $P$, $Q$ - two discrete probability distributions:

$$D_{KL}(P\|Q) = \sum_{i=1}^{N} P(x_i) \log \frac{P(x_i)}{Q(x_i)}$$

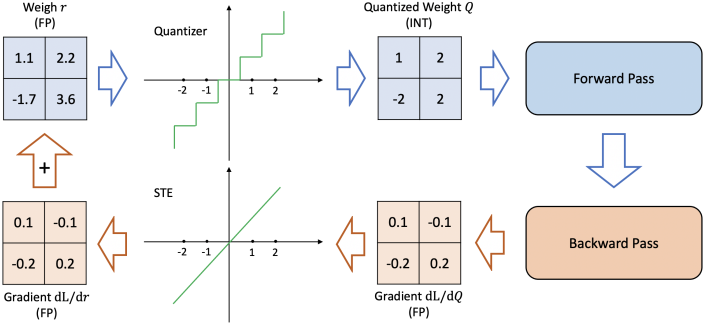
- Intuition: KL divergence measures the amount of information lost when approximating a given encoding.

# Quantization Aware Training (QAT)

# QAT: Weight

## Straight Through Estimator (STE)[4]

- Forward integer, Backward floating point
- Rounding to nearest



---

[4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville (2013). "Estimating or propagating gradients through stochastic neurons for conditional computation". In: *arXiv preprint arXiv:1308.3432*.

# Better Gradients

## Is Straight-Through Estimator (STE) the best?

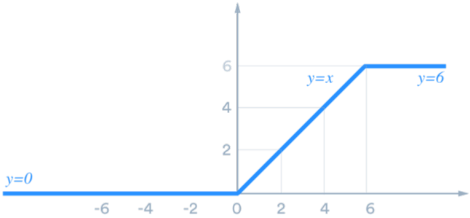
- Gradient mismatch: the gradients of the weights are not generated using the value of weights, but rather its quantized value.
- Poor gradient: STE fails at investigating better gradients for quantization training.

# QAT: Activation

## PArameterized Clipping acTivation (PACT)[5]

- Relu6 → clipping
- threshold → clipping range in quantization
- range upper/lower bound trainable

$$y = PACT(x) = 0.5(|x| - |x-\alpha| + \alpha) = \begin{cases} 0, & x \in (-\infty, 0) \\ x, & x \in [0, \alpha) \\ \alpha, & x \in [\alpha, +\infty) \end{cases}$$



---

[5] Jungwook Choi, Zhuo Wang, et al. (2018). "Pact: Parameterized clipping activation for quantized neural networks". In: *arXiv preprint arXiv:1805.06085*.

# PArameterized Clipping acTivation Function (PACT)[6]

- A new activation quantization scheme in which the activation function has a parameterized clipping level $\alpha$.

- The clipping level is dynamically adjusted vias stochastic gradient descent (SGD)-based training with the goal of minimizing the quantization error.

- In PACT, the convolutional ReLU activation function in CNN is replaced with:

$$f(x) = 0.5\left(|x| - |x-\alpha| + \alpha\right) = \begin{cases} 0, & x \in (\infty, 0) \\ x, & x \in [0, \alpha) \\ \alpha, & x \in [\alpha, +\infty) \end{cases}$$

where $\alpha$ limits the dynamic range of activation to $[0, \alpha]$.

---

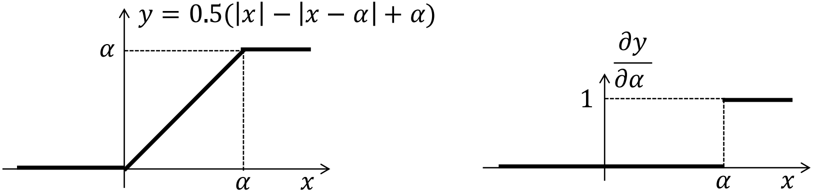
[6] Jungwook Choi, Swagath Venkataramani, et al. (2019). “Accurate and efficient 2-bit quantized neural networks”. In: *Proceedings of Machine Learning and Systems* 1.

# PArameterized Clipping acTivation Function (PACT)

- The truncated activation output is the linearly quantized to $k$-bits for the dot-product computations:

$$y_q = \text{round}\left(y \cdot \frac{2^k - 1}{\alpha}\right) \cdot \frac{\alpha}{2^k - 1}$$

- With this new activation function, $\alpha$ is a variable in the loss function, whose value can be optimized during training.
- For back-propagation, gradient $\frac{\partial y_q}{\partial \alpha}$ can be computed using STE to estimate $\frac{\partial y_q}{\partial y}$ as 1.



PACT activation function and its gradient.