Data Precision and Quantization

Kang Zhao

zhaokang@bupt.edu.cn

Outline

- **Floating-point and fixed-point representations**
- **Fixed-point in Vitis HLS**
- **Data quantization in Machine Learning**

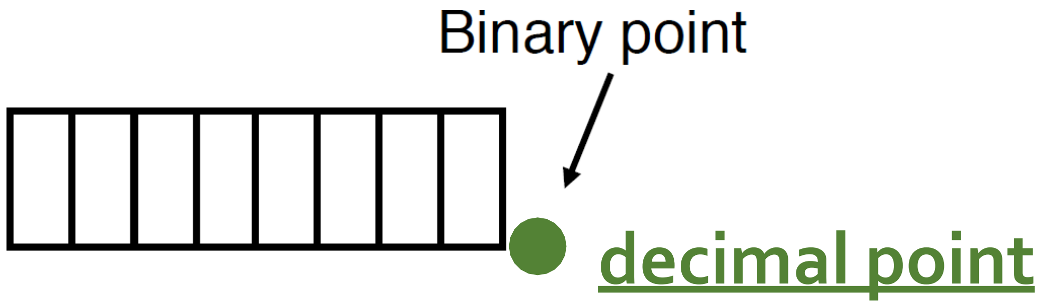
Binary Number Representation

Unsigned number

- **MSB has weight 2^{n-1}**

Signed number

- **MSB has weight -2^{n-1}**

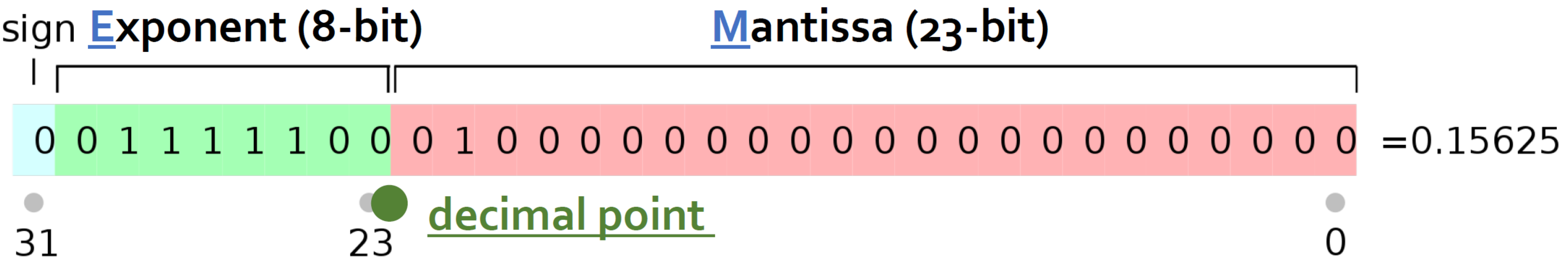


Examples: assuming integers here

2^3	2^2	2^1	2^0	unsigned
1	0	1	1	= 11

-2^3	2^2	2^1	2^0	2'c
1	0	1	1	= -5

IEEE 754 Floating Point Standard



$$\text{Value} = (-1)^{s} \times 2^{E-127} \times (1.M)$$
$$= 1 \times 2^{124-127} \times (1 + (0.01)_2)$$
$$= 1.25 \times 0.125$$
$$= 0.15625$$

Fixed-Point Representation

- What if we fix the decimal point? → Fixed-point numbers

2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	unsigned
1	0	1	1	●	0	1	11.25

decimal point

2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2'C
1	0	1	1	●	0	1	?

Fixed-Point Representation

- What if we fix the decimal point? → Fixed-point numbers

2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	unsigned
1	0	1	1	●	0	1	11.25

decimal point

2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2'C
1	0	1	1	●	0	1	-4.75 (-8 + 3.25)

Fixed-Point Overflow and Underflow

- **Overflow: happens at MSB**
  - When a number is larger than the largest that can be represented

2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	
0	0	1	0	1	1	0	1	0	0	$= 11.25$
	0	1	0	1	1	0	1	0	0	$= 11.25$
		1	0	1	1	0	1	0	0	$= 11.25$
	Drop MSB		0	1	1	0	1	0	0	$= 3.25$

- **Underflow: happens at LSB**
  - When a number is smaller than the smallest that can be represented

Wrapping and Rounding

- **Wrapping is one common (& efficient) way of handling overflow: drop the MSBs of the original number**
  - But… may result in a totally wrong number

-2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	two's complement
1	0	1	1	0	1	0	0	$= -4.75$

-2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	two's complement
0	1	1	0	1	0	0	$= 3.25$

Wrapping and Rounding

- **Rounding: when a number cannot be represented precisely in a given number of fractional bits (also called quantization)**
- **Rounding down (round to negative infinity): drop the extra fractional bits (LSBs)**
  - Result in numbers that are more negative

$$
\begin{aligned}
0b0100.00 &= 4.0 \\
0b0011.11 &= 3.75 \\
0b0011.10 &= 3.5 \\
0b0011.01 &= 3.25 \\
0b0011.00 &= 3.0 \\
0b1100.00 &= -4.0 \\
0b1011.11 &= -4.25 \\
0b1011.10 &= -4.5 \\
0b1011.01 &= -4.75 \\
0b1011.00 &= -5.0
\end{aligned}
\rightarrow \begin{array}{l}\text{Round to}\\ \text{Negative}\\ \text{Infinity}\end{array} \rightarrow
\begin{aligned}
0b0100.0 &= 4.0 \\
0b0011.1 &= 3.5 \\
0b0011.1 &= 3.5 \\
0b0011.0 &= 3.0 \\
0b0011.0 &= 3.0 \\
0b1100.0 &= -4.0 \\
0b1011.1 &= -4.5 \\
0b1011.1 &= -4.5 \\
0b1011.0 &= -5.0 \\
0b1011.0 &= -5.0
\end{aligned}
$$

Wrapping and Rounding

- **Rounding to nearest even: always picks the nearest representable number**
  - Minimizes rounding errors
  - Error tends to **cancel out** when computing sums – **unbiased rounding**

0b0100.00	$= 4.0$		0b0100.0	$= 4.0$	
0b0011.11	$= 3.75$		0b0100.0	$= 4.0$	**Goes larger**
0b0011.10	$= 3.5$		0b0011.1	$= 3.5$	
0b0011.01	$= 3.25$		0b0011.0	$= 3.0$	**Goes smaller**
0b0011.00	$= 3.0$	$\rightarrow$ Round to Nearest Even $\rightarrow$	0b0011.0	$= 3.0$	
0b1100.00	$= -4.0$		0b1100.0	$= -4.0$	
0b1011.11	$= -4.25$		0b1100.0	$= -4.0$	
0b1011.10	$= -4.5$		0b1011.1	$= -4.5$	
0b1011.01	$= -4.75$		0b1011.0	$= -5.0$	
0b1011.00	$= -5.0$		0b1011.0	$= -5.0$	

Outline

- Floating-point and fixed-point representations
- Fixed-point in Vitis HLS
- Data quantization in Machine Learning

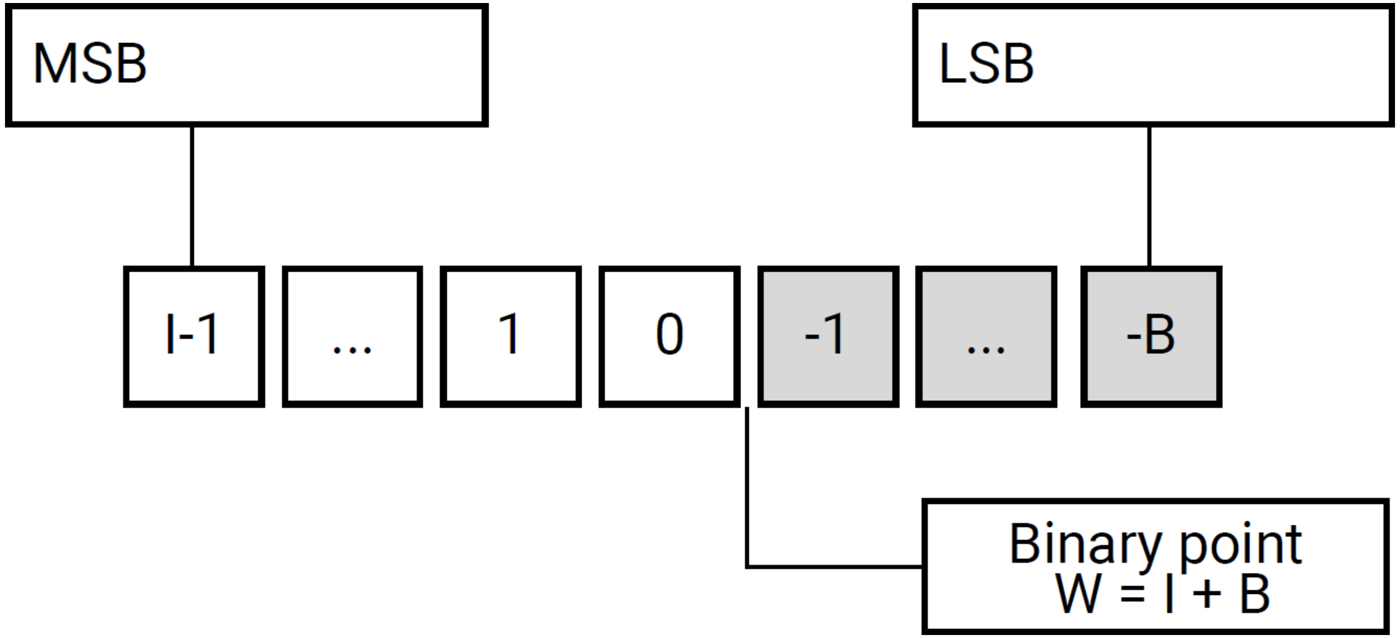
Arbitrary-Precision Integer in HLS

- **C/C++ only provides a limited set of native integer types**
  - char (8b), short (16b), int (32b), long (?), long long (64b)
- **Arbitrary precision integer in HLS**
  - Signed: **ap_int**; Unsigned **ap_uint**
  - Templatized class ap_int<W> or ap_uint<W>
    - W is the user-specified bitwidth

```
#include “ap_int.h”
…
ap_int<9>      x;  // 9-bit
ap_uint<17>    y;  // 17-bit unsigned
ap_uint<512>   z;  // 512-bit unsigned
```

Arbitrary-Precision Fixed-Point in Vitis HLS

- **Signed: ap_fixed; Unsigned ap_ufixed**
  - Templatized class ap_fixed<W, I, Q, O>
  - Q: quantization mode; O: overflow mode



Example: Fixed-Point Modeling

- `ap_ufixed<11, 8, AP_TRN, AP_WRAP>   x;`

b7	b6	b5	…	b1	b0	…	b-3

  - **11** is the total number of bits in the type
  - **8** bits is the integer bitwidth
  - **AP_TRN** defines **truncation** behavior for **quantization**
  - **AP_WRAP** defines **wrapping** behavior for **overflow**

Quantization and Overflow Modes

AP_RND	Round to plus infinity
AP_RND_ZERO	Round to zero
AP_RND_MIN_INF	Round to minus infinity
AP_RND_INF	Round to infinity
AP_RND_CONV	Convergent rounding
AP_TRN	Truncation to minus infinity (default)
AP_TRN_ZERO	Truncation to zero

Quantization and Overflow Modes

AP_SAT[1]	Saturation
AP_SAT_ZERO[1]	Saturation to zero
AP_SAT_SYM[1]	Symmetrical saturation
AP_WRAP	Wrap around (default)
AP_WRAP_SM	Sign magnitude wrap around

AP_SAT Requires extra complex logic

```
ap_fixed<4, 4, AP_RND, AP_SAT> UAPFixed4 = 19.0; // Yields: 7.0
ap_fixed<4, 4, AP_RND, AP_SAT> UAPFixed4 = -19.0; // Yields: -8.0
ap_ufixed<4, 4, AP_RND, AP_SAT> UAPFixed4 = 19.0; // Yields: 15.0
ap_ufixed<4, 4, AP_RND, AP_SAT> UAPFixed4 = -19.0; // Yields: 0.0
```

Pros and Cons of Fixed Point

- **Pros**
  - Low memory, low resource (DSP, peripheral logic), low latency
- **Cons**
  - Uncertain accuracy

- **Vitis HLS can also synthesize floating point**
  - Requires significant amount of computation → a large amount of resource usage and many cycles of latency
  - Floating point numbers should be avoided unless absolutely necessary

Backup

Outline

- **Floating-point and fixed-point representations**
- **Fixed-point in Vitis HLS**
- **Data quantization in Machine Learning**

Quantization for Machine Learning

- **Why do we care?**
  - Faster, smaller, easier!

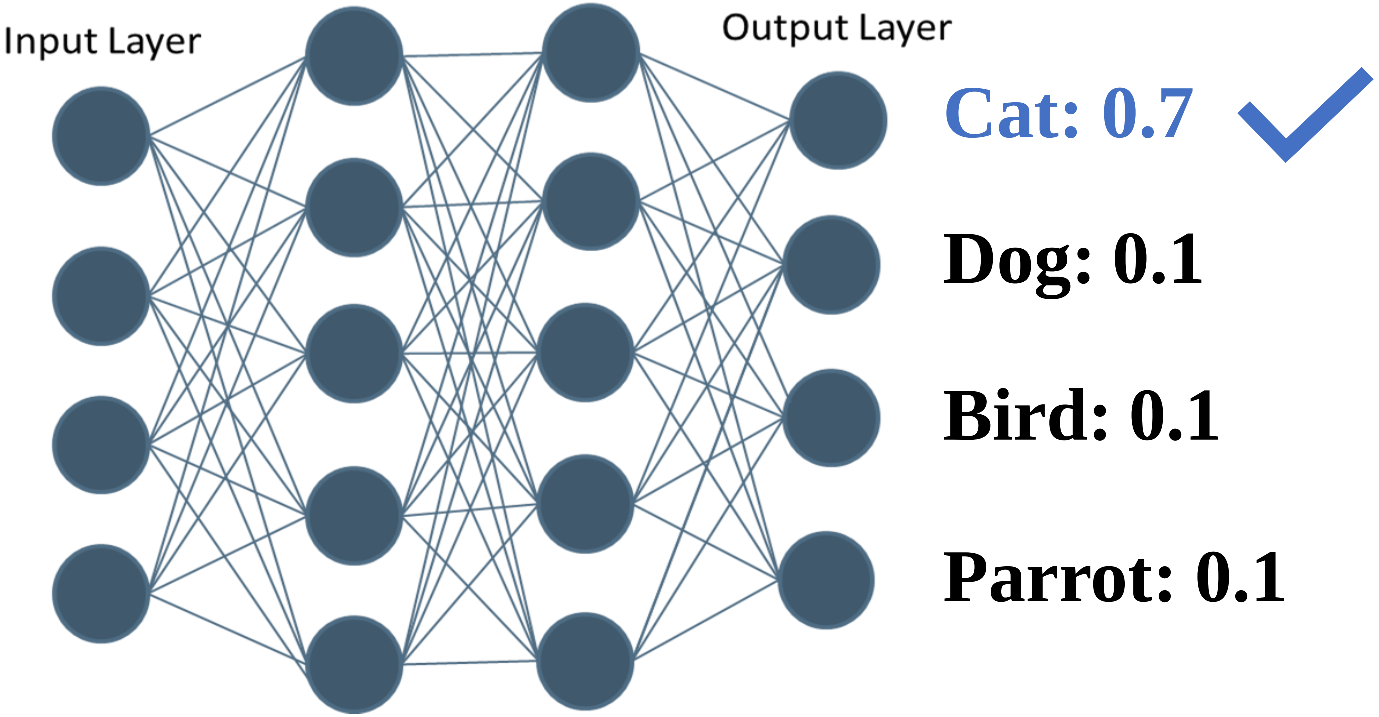
- **Why does it work?**

- **Quantization Methods**
  - Fixed-point
  - Integer

Reference: https://sahnimanas.github.io/post/quantization-in-tflite/

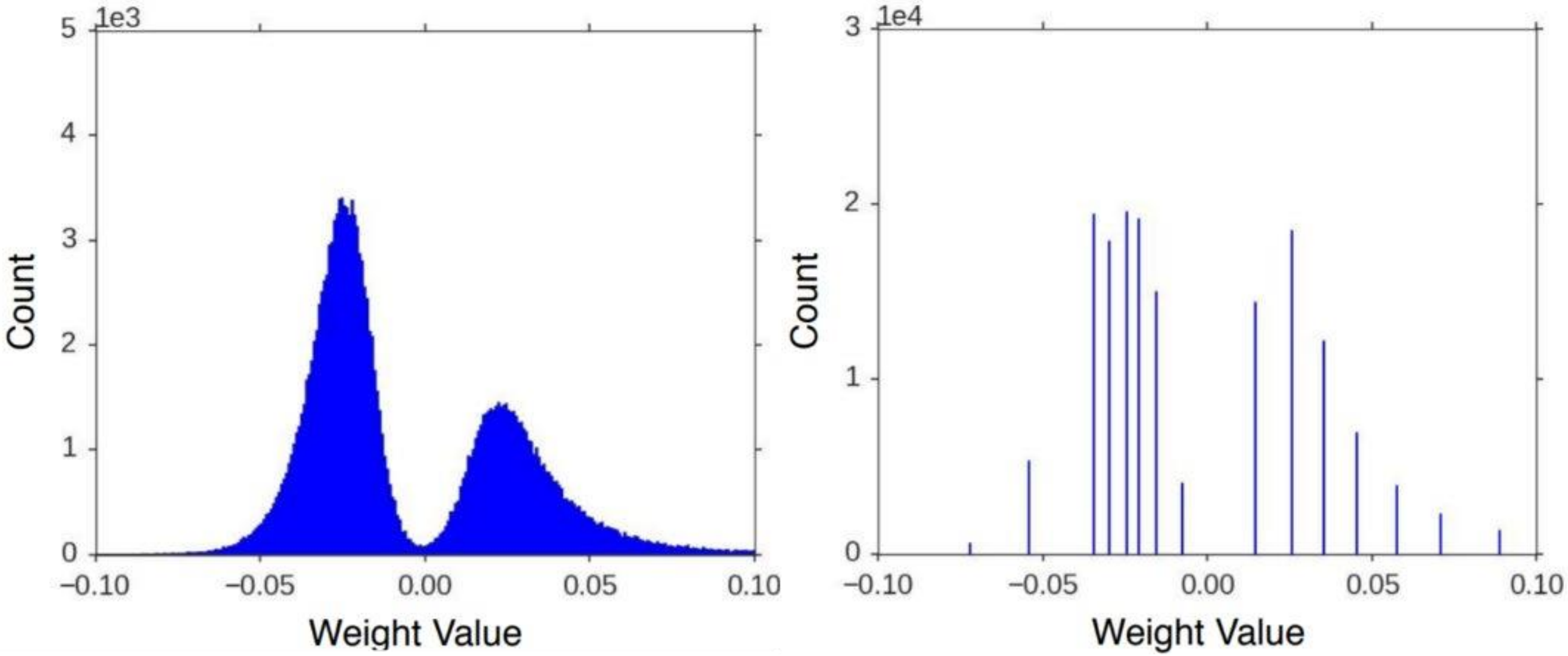
Why Does Quantization Work for ML?

- **DNNs are known to be robust to noise and small perturbations once trained**
  - 80% chance to be a cat? 75% chance to be a cat? Still a cat!



Why Does Quantization Work for ML?

- **DNNs are known to be robust to noise and small perturbations once trained**
  - 80% chance to be a cat? 75% chance to be a cat? Still a cat!
- **Weights and activations often tend to lie in a small range**
  - Can be estimated beforehand



Quantization Method 1 – Fixed Point

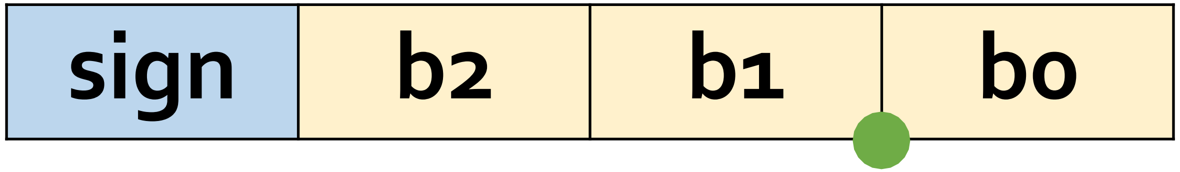
- **Using fixed-point instead of floating point is a convenient and common approach on FPGA**

```
#include “ap_fixed.h”

Typedef ap_fixed<9, 3, AP_TRN, AP_WRAP> my_fixed_type;

my_fixed_type weights[dim1][dim2];
my_fixed_type bias[dim1];
…
weights[0][0] = (my_fixed_type) weights_floating[0][0];
…
```

Quantization Method 1 – Fixed Point

- Using fixed-point instead of floating point is a convenient and common approach on FPGA
  - Need to determine (sometimes trial-and-error) the W and I
  - W: total number of bits
  - I: number of integer bits

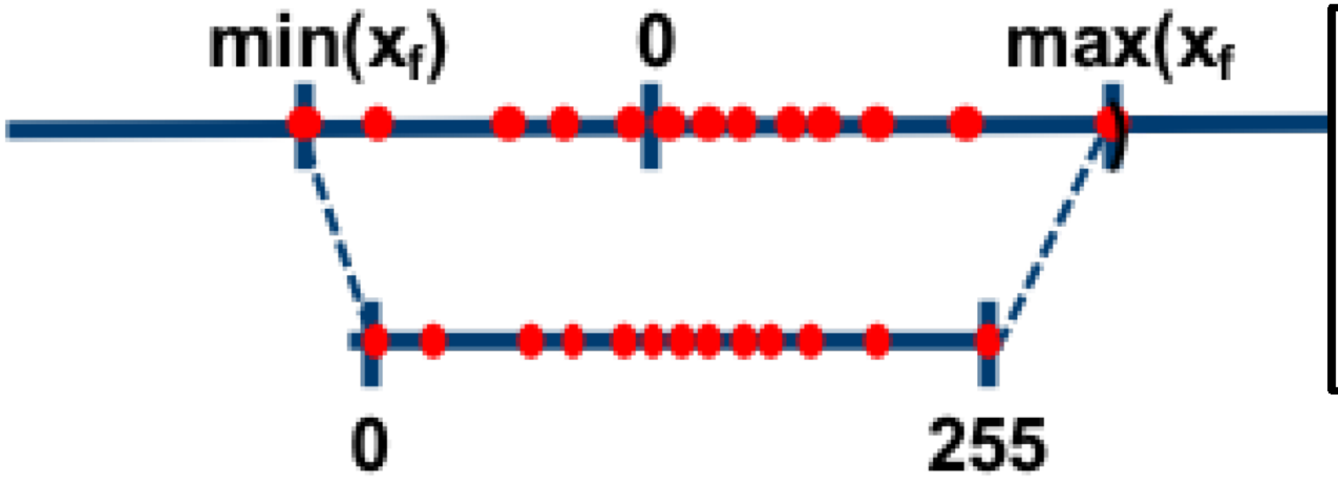
- **Drawback**: doesn't work well for low bit-width, e.g., 8-bit
  - The range of the represented number is strictly limited by W and I



-4 to 3 0.5

Quantization Method 2 – Integer

- With a scaling and a bias (shift), can represent a much larger range



8-bit

$$f = \frac{f_{max} - f_{min}}{2^B - 1} \times q + bias$$

$$f = scale \times q + bias$$

- f : original floating-point value
- q : quantized integer
- *scale* and *bias*: can be floating or fixed point

Quantization Method 2 – Integer (Example)

-257.8	-117.2	-6.3	0	18.4	25.3	66.9	77.3	128.1	145.6

3-bit: $2^3 = 8$

0	1	2	3	4	5	6	7

Quantization Method 2 – Integer (Example)

-257.8	-117.2	-6.3	0	18.4	25.3	66.9	77.3	128.1	145.6

$$\frac{145.6 - (-257.8)}{2^3 - 1} = 57.63$$

Scaling factor

3-bit: $2^3 = 8$

0	1	2	3	4	5	6	7

Quantization Method 2 – Integer (Example)

-257.8	-117.2	-6.3	0	18.4	25.3	66.9	77.3	128.1	145.6

$\div\, 2211.1133$

-4.47	-2.03	-0.11	0.00	0.32	0.44	1.16	1.34	2.22	2.53

Next is to shift

3-bit: $2^3 = 8$

0	1	2	3	4	5	6	7

Quantization Method 2 – Integer (Example)

-257.8	-117.2	-6.3	0	18.4	25.3	66.9	77.3	128.1	145.6

$\div 2211.1133$

-4.47	-2.03	-0.11	0.00	0.32	0.44	1.16	1.34	2.22	2.53

$+4.47$ (is this the best choice?)

3-bit: $2^3 = 8$

0	1	2	3	4	5	6	7

Quantization Method 2 – Integer (Example)

-257.8	-117.2	-6.3	0	18.4	25.3	66.9	77.3	128.1	145.6

$\div 2211.1133$

-4.47	-2.03	-0.11	0.00	0.32	0.44	1.16	1.34	2.22	2.53

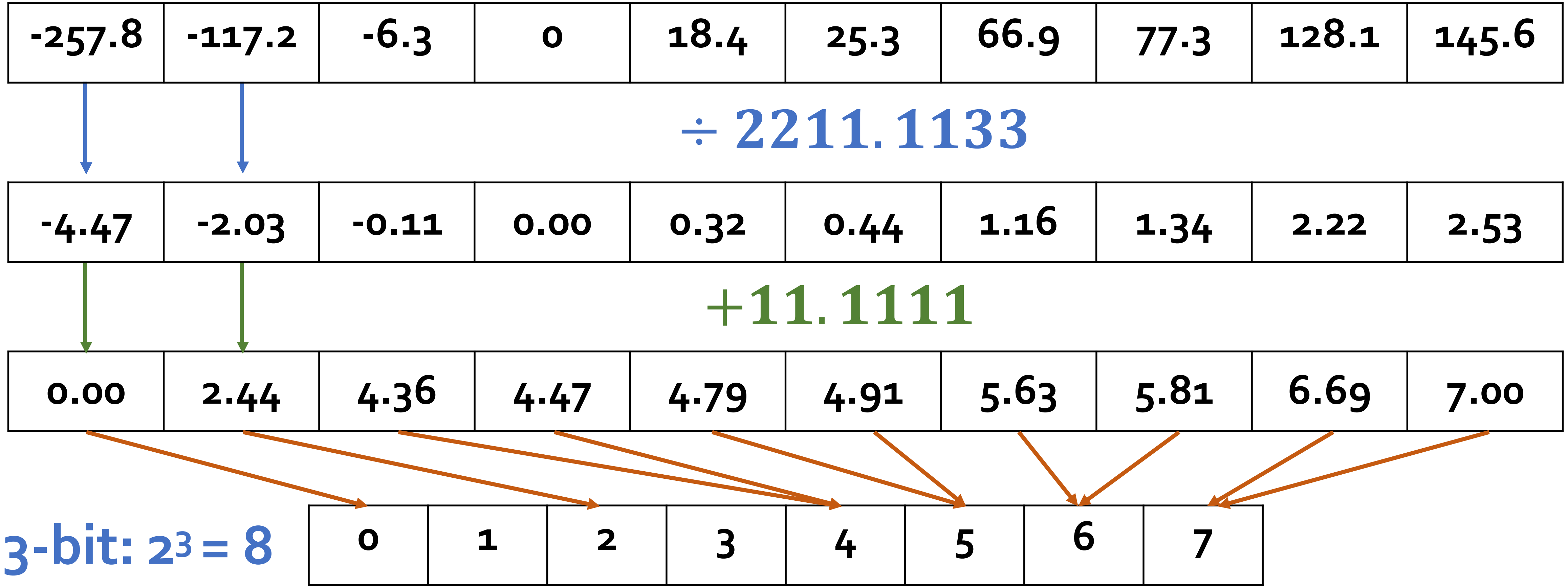
$+11.1111$

0.00	2.44	4.36	4.47	4.79	4.91	5.63	5.81	6.69	7.00

3-bit: $2^3 = 8$

0	1	2	3	4	5	6	7

Quantization Method 2 – Integer (Example)



Quantization Method 2 – Integer (Example)

-257.8	-117.2	-6.3	0	18.4	25.3	66.9	77.3	128.1	145.6

$\div 2211.1133$

-4.47	-2.03	-0.11	0.00	0.32	0.44	1.16	1.34	2.22	2.53

$+11.1111$

0.00	2.44	4.36	4.47	4.79	4.91	5.63	5.81	6.69	7.00

3-bit: $2^3 = 8$

0	1	2	3	4	5	6	7

Zeros are very common in ML but are tampered

Quantization Method 2 – Integer (Example)

-257.8	-117.2	-6.3	0	18.4	25.3	66.9	77.3	128.1	145.6

$\div\ 57.63$

-4.47	-2.03	-0.11	0.00	0.32	0.44	1.16	1.34	2.22	2.53

$+5$

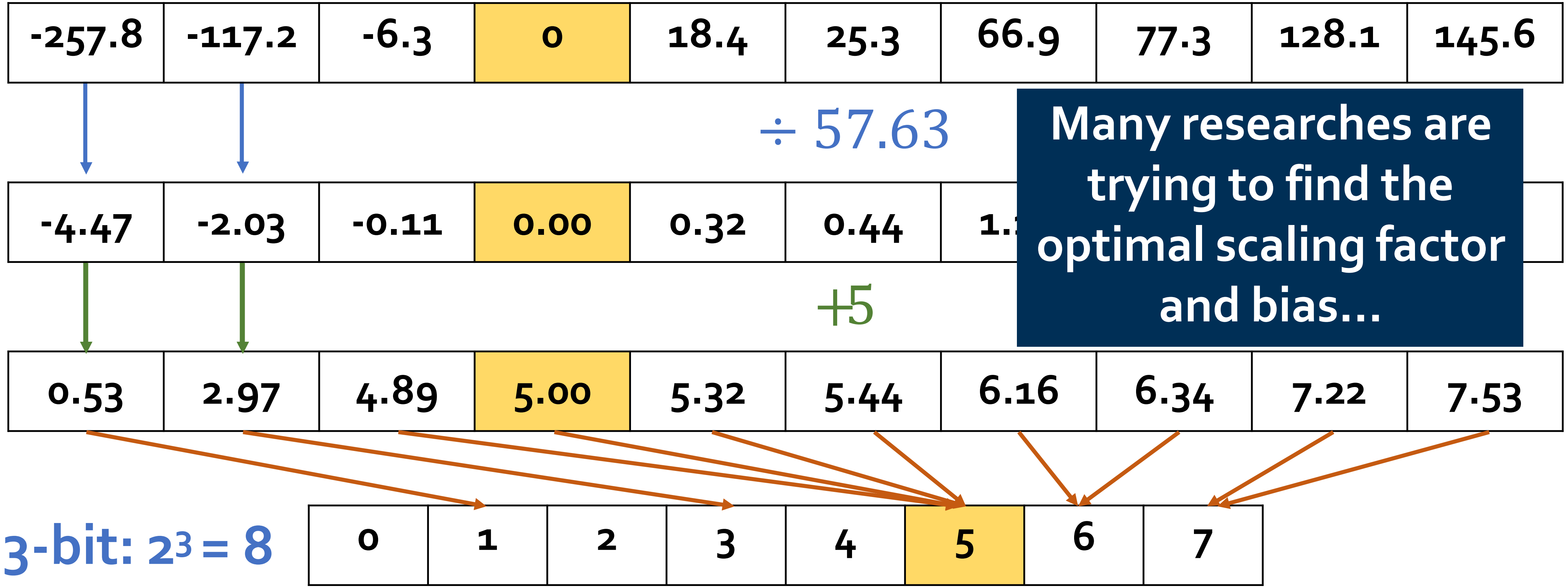
0.53	2.97	4.89	5.00	5.32	5.44	6.16	6.34	7.22	7.53

3-bit: $2^3 = 8$

0	1	2	3	4	5	6	7

A better approach: align the zero!

Quantization Method 2 – Integer (Example)



A better approach: align the zero!