Numeric Data Types

CS 6501: AI Systems

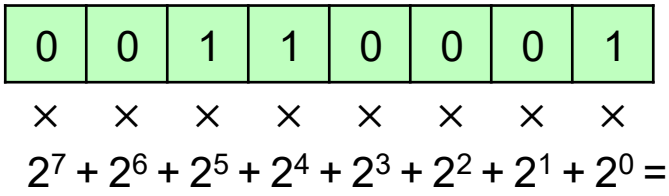
Fall 2026

Lecture 2

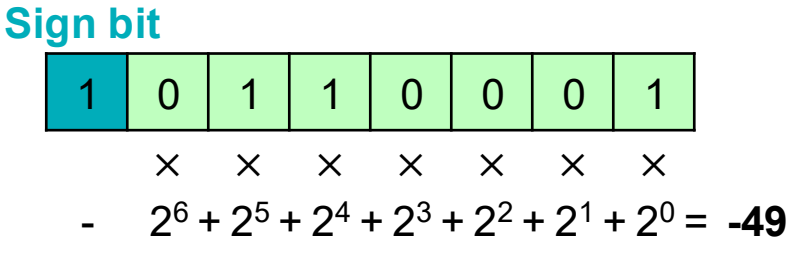
Yue Cheng

Integer

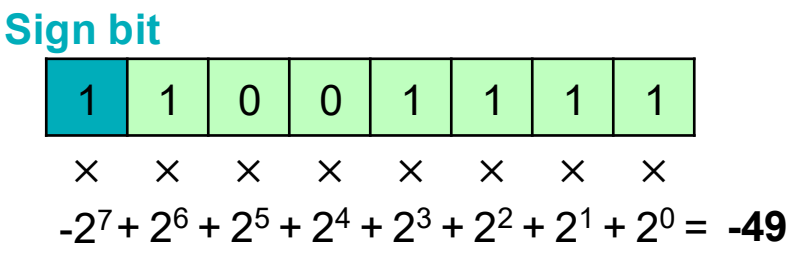
- Unsigned integer
  - *n*-bit range: $[0, 2^n - 1]$



- Signed integer
  - Sign-magnitude representation
  - *n*-bit range: $[-2^{n-1} + 1, 2^{n-1} - 1]$
  - Both 000…00 and 100…00 represent 0



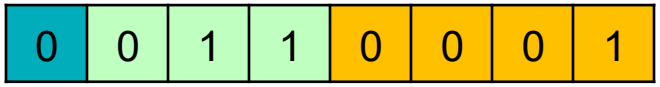
  - Two's complement representation
  - n-bit range: $[-2^{n-1}, 2^{n-1} - 1]$
  - 000…00 represents 0
  - 100…00 represents -2^{n-1}



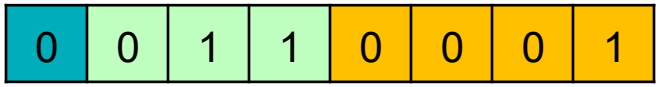
Fixed-point number

Integer . Mantissa

“Decimal” point



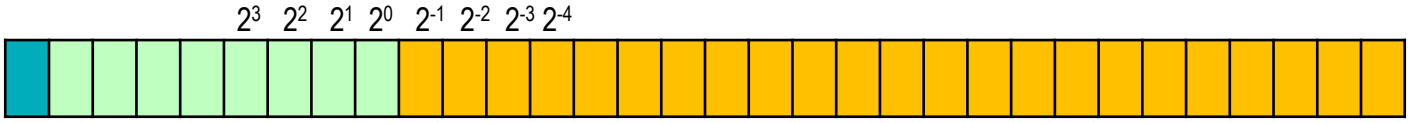
$$-2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} = \mathbf{3.0625}$$



$$\left(-2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0\right) \times 2^{-4} = \mathbf{49} \times \mathbf{0.0625} = \mathbf{3.0625}$$

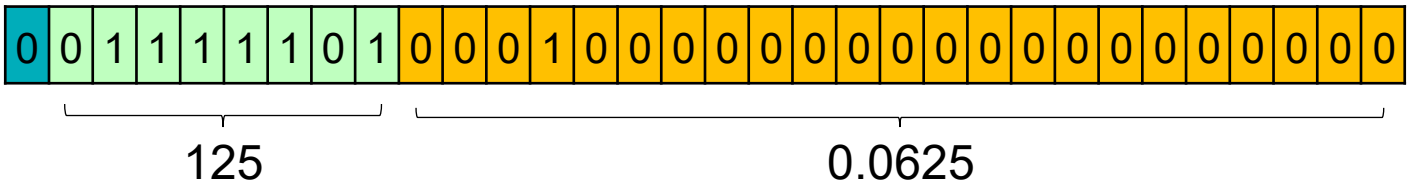
Floating-point number

Example: 32-bit floating-point number in IEEE 754



$$(-1)^{\text{sign}} \times (1 + \text{mantissa}) \times 2^{\text{exponent} - 127}$$ ← **Exponent bias** = 127 = 2^{8-1} -1

$$1.0625 \times 2^{-2} = (1 + \underline{0.0625}) \times 2^{\underline{125}-127} = \mathbf{0.265625}$$



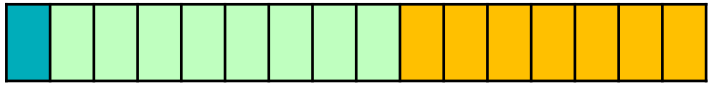
Floating-point number

Exponent width → Range; Mantissa width → Precision

IEEE 754 single precision 32-bit float (IEEE FP32)

IEEE 754 half precision 16-bit float (IEEE FP16)

Google brain float (BF16)



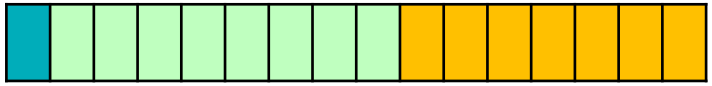
Floating-point number

Exponent width → Range; Mantissa width → Precision

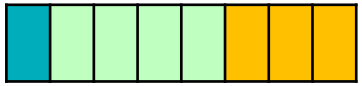
IEEE 754 single precision 32-bit float (IEEE FP32)

IEEE 754 half precision 16-bit float (IEEE FP16)

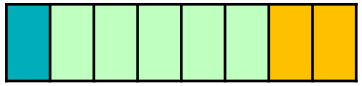
Google brain float (BF16)



NVIDIA FP8 (E4M3)



NVIDIA FP8 (E5M2) for gradient in the backward



INT4 and FP4

Exponent width → Range; Mantissa width → Precision

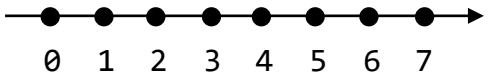
INT4

S			
0	0	0	1
0	1	1	1

=1

=7

-1,-2,-3,-4,-5,-6,-7,-8
0, 1, 2, 3, 4, 5, 6, 7



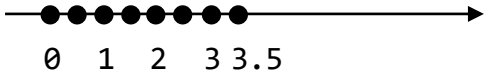
FP4 (E1M2)

S	E	M	M
0	0	0	1
0	1	1	1

=$0.25\times2^{1-0}$=0.5

=$(1+0.75)\times2^{1-0}$=3.5

-0,-0.5,-1,-1.5,-2,-2.5,-3,-3.5
0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5



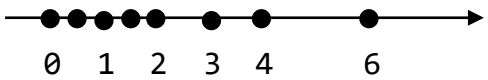
FP4 (E2M1)

S	E	E	M
0	0	0	1
0	1	1	1

=$0.25\times2^{1-1}$=0.5

=$(1+0.5)\times2^{3-1}$=6

-0,-0.5,-1,-1.5,-2,-3,-4,-6
0, 0.5, 1, 1.5, 2, 3, 4, 6



no inf, no NaN

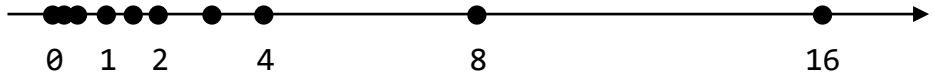
FP4 (E3M0)

S	E	E	E
0	0	0	1
0	1	1	1

=$(1+0)\times2^{1-3}$=0.25

=$(1+0)\times2^{7-3}$=16

-0,-0.25,-0.5,-1,-2,-4,-8,-16
0, 0.25, 0.5, 1, 2, 4, 8, 16



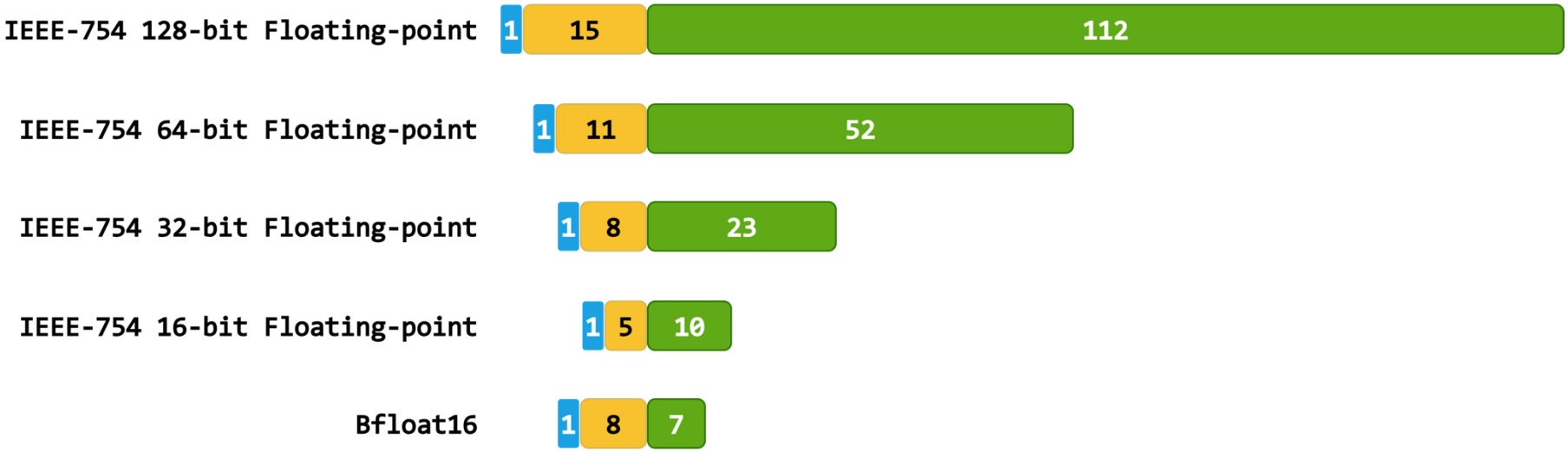
no inf, no NaN

Python numeric types (built in)

https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex

Python numeric types

- int
  - No max/min size (Python is unusual in this way)
  - Bigger values -> more bits necessary
- float
  - Defaults 64 bits (double precision)
    - You can also use float32 given a certain framework (e.g., PyTorch, etc.)
  - Historically, pre-trained ML models use **float32** for parameters
    - For modern AI (LLMs), **bfloat16** is common



*: https://docs.nvidia.com/cuda/cuda-programming-guide/05-appendices/mathematical-functions.html

So, does the order matter?

$$a = 0.1$$
$$b = 1e20$$
$$c = -1e20$$

$$(a + b) + c = ?$$
$$a + (b + c) = ?$$

Where did 0.1 go?