Floating-Point, Fixed-Point, and Integer Number Representation Formats

Sparsh Mittal

Solved Examples

Floating-Point Example

- Represent –0.75
  - $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$
  - S = 1
  - Fraction = $1000\ldots00_2$
  - Exponent = –1 + Bias
    - Single: $-1 + 127 = 126 = 01111110_2$
    - Double: $-1 + 1023 = 1022 = 01111111110_2$
- Single: 1011111101000…00
- Double: 1011111111101000…00

Floating-Point Example

- What number is represented by the single-precision float
  11000000101000...00
  - S = 1
  - Fraction = $01000...00_2$
  - Exponent = $10000001_2 = 129$
- $x = (-1)^1 \times (1 + .01_2) \times 2^{(129-127)}$
  $= (-1) \times 1.25 \times 2^2$
  $= -5.0$

Single-Precision Range

- Exponents 00000000 and 11111111 reserved
- Smallest value
  - Exponent: 00000001
    $\Rightarrow$ actual exponent = 1 – 127 = –126
  - Fraction: 000…00 $\Rightarrow$ significand = 1.0
  - $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$
- Largest value
  - exponent: 11111110
    $\Rightarrow$ actual exponent = 254 – 127 = +127
  - Fraction: 111…11 $\Rightarrow$ significand $\approx$ 2.0
  - $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

Double-Precision Range

- Exponents 0000…00 and 1111…11 reserved
- Smallest value
  - Exponent: 00000000001
    $\Rightarrow$ actual exponent = 1 – 1023 = –1022
  - Fraction: 000…00 $\Rightarrow$ significand = 1.0
  - $\pm 1.0 \times 2^{-1022} \approx \pm 2.2 \times 10^{-308}$
- Largest value
  - Exponent: 11111111110
    $\Rightarrow$ actual exponent = 2046 – 1023 = +1023
  - Fraction: 111…11 $\Rightarrow$ significand $\approx$ 2.0
  - $\pm 2.0 \times 2^{+1023} \approx \pm 1.8 \times 10^{+308}$

Why do we use bias?

Let us compute the range of single-precision number, without using bias.

Single-Precision Range **[if no bias is used]**

- Smallest value
  - Exponent: 00000001
    $\Rightarrow$ actual exponent = $1 - 0 = 1$
  - Fraction: 000…00 $\Rightarrow$ significand = 1.0
  - $\pm 1.0 \times 2^{1} \approx \pm 2$
- Largest value
  - exponent: 11111110
    $\Rightarrow$ actual exponent = $254\text{-}0 = 254$
  - Fraction: 111…11 $\Rightarrow$ significand $\approx$ 2.0
  - $\pm 2.0 \times 2^{+254} \approx \pm 2.9 \times 10^{+76}$

1. On not using bias, we can only represent large numbers, and not small numbers

Why do we use bias?

To represent small numbers, we can start using negative values. But that would be confusing

Use of a bias means, which store excess exponent, which is always positive.

Denormal Numbers

- Smallest +ve normal number : 2^{-126}
- Smallest denormal number :
  - $0.00..01 * 2^{-126} = (2^{-23})*2^{-126}$

    $=2^{-149}$

- Largest denormal number :
  - $0.11...11 * 2^{-126} = (1 – 2^{-23})*2^{-126}$
  - $=2^{-126} - 2^{-149}$

- For positive denormal numbers, the range is $[2^{-149} , 2^{-126} – 2^{-149}]$

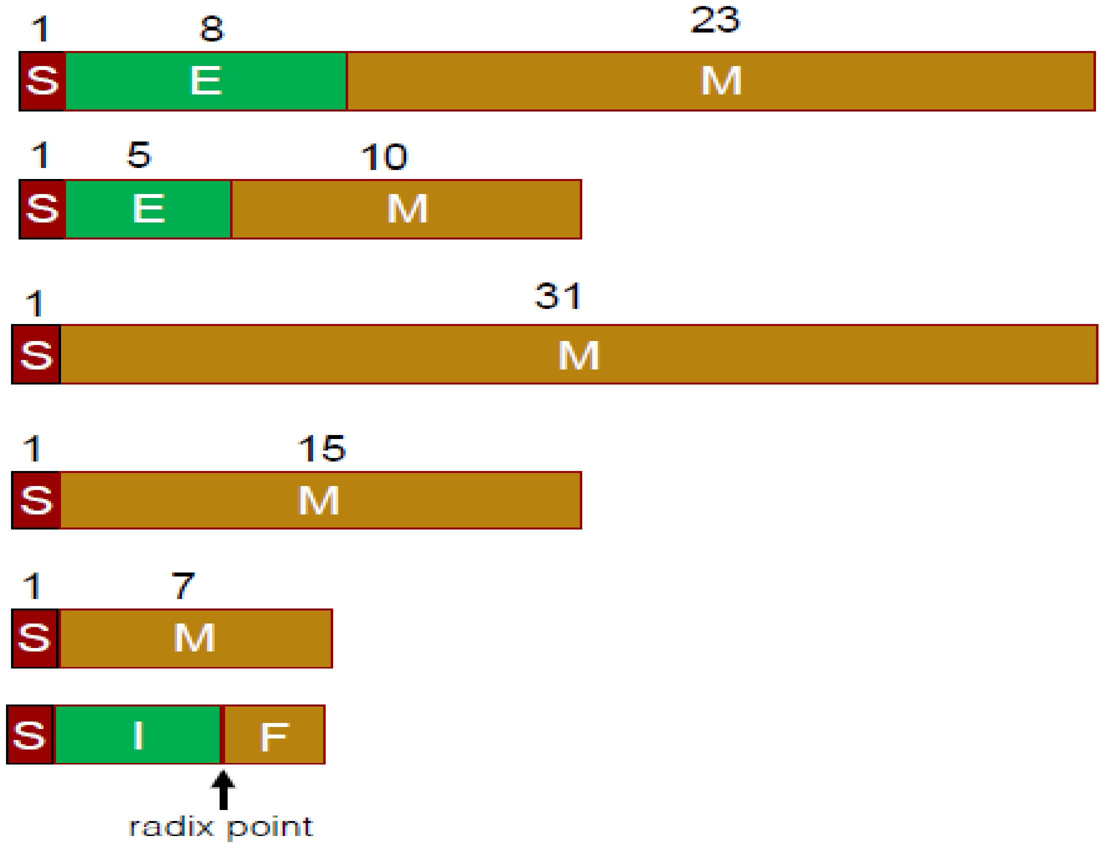
Solved example on Fixed Point Number

* In an application, all data values are positive. The highest value we need to store is 36000. We want to use 20-bit storage and fixed-point number system. How many bits should we allocate for integer and fraction part, so we can have as high precision as possible, with no overflow.
* Solution: Since all data values are positive, no sign bit is needed. Now, $2^n-1>=36000$
* $n >= \log_2(36001) \rightarrow n >= 15.14$
* Integer part: 16 bits, Fraction Part: 4 bits

Comparison Between Floating-Point (FP), Fixed-Point(FxP), and Integer

Quick Summary

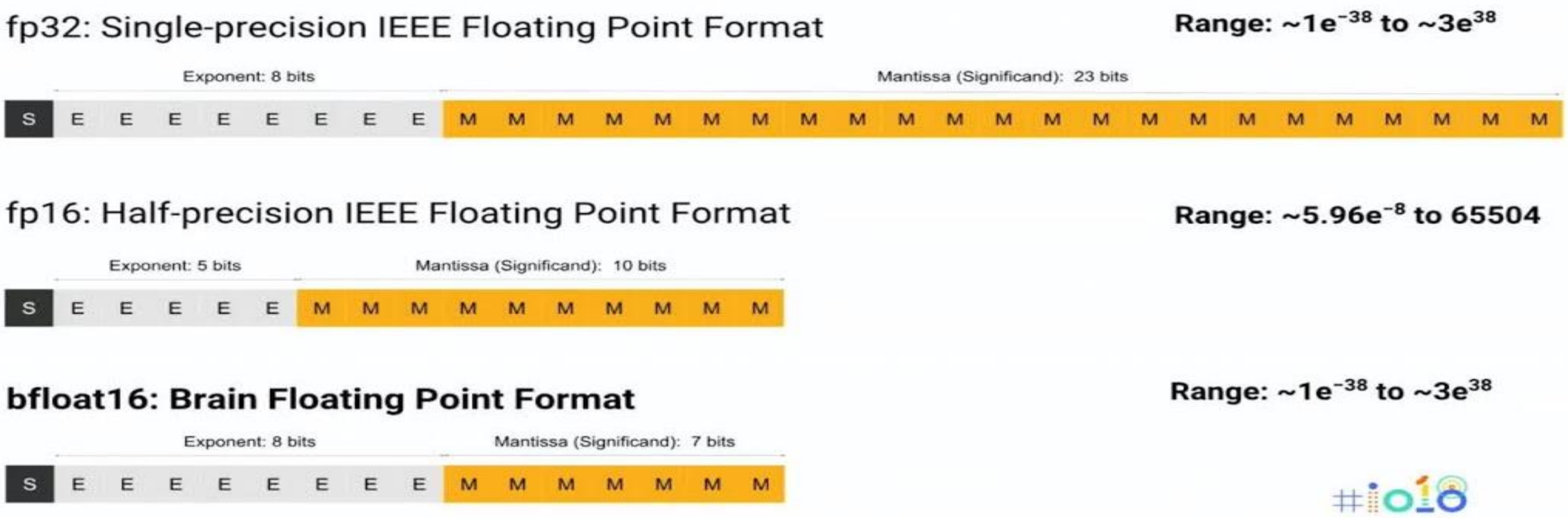
$$(-1)^{S} \times (1.M) \times 2^{E}$$



	Range	Error
FP32	10^{-38} - 10^{38}	.000006%
FP16	6×10^{-5} - 6×10^{4}	.05%
Int32	$0 - 2\times10^{9}$	½
Int16	$0 - 6\times10^{4}$	½
Int8	0 – 127	½
Fixed point	-	-

Dally, High Performance Hardware for Machine Learning, NIPS'2015

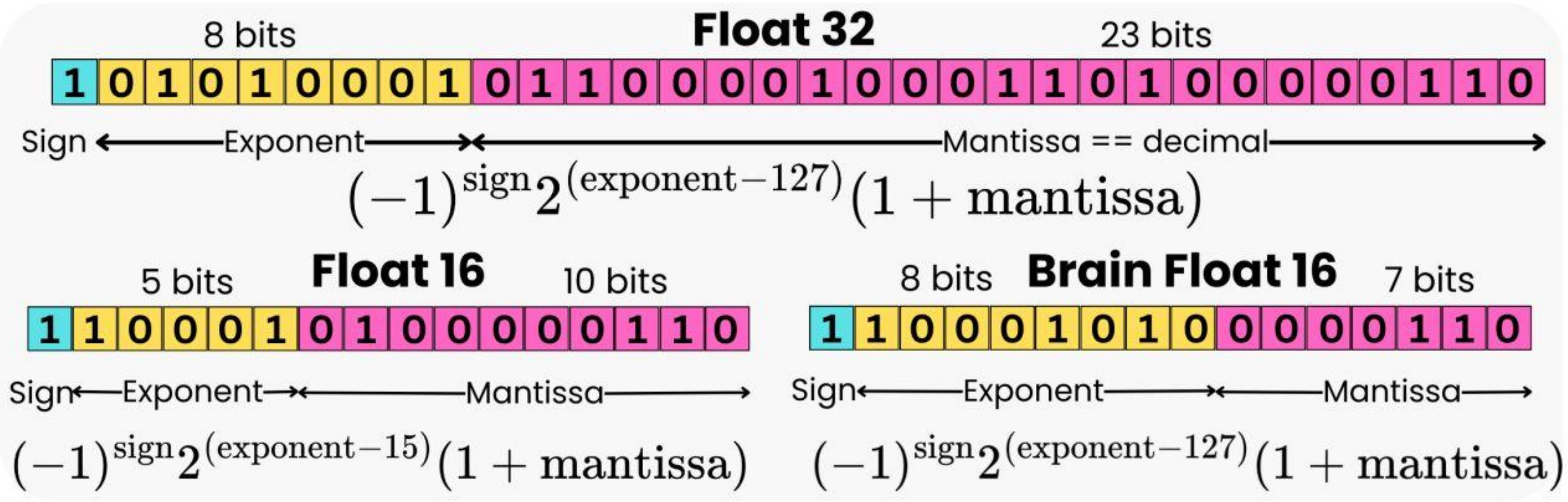
Guide to Floating Point Formats



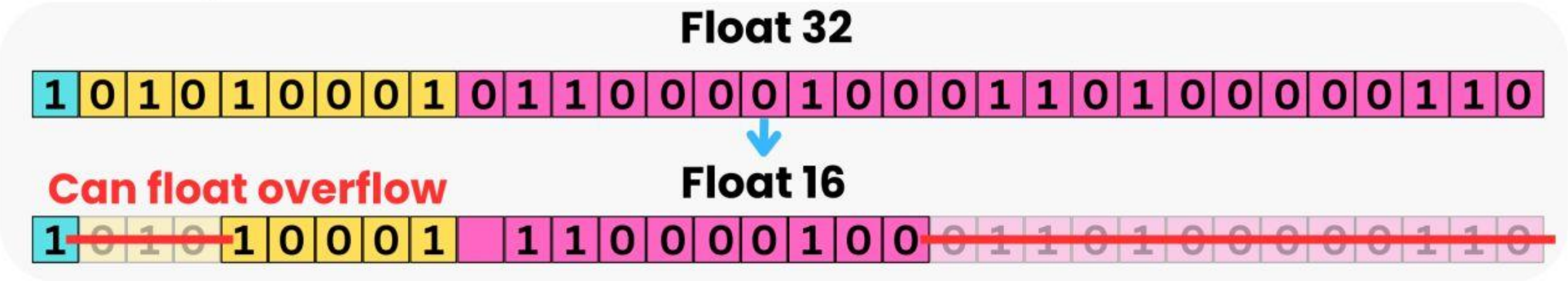
Why is BFloat16 Better than Float16 for Model Training

TheAiEdge.io

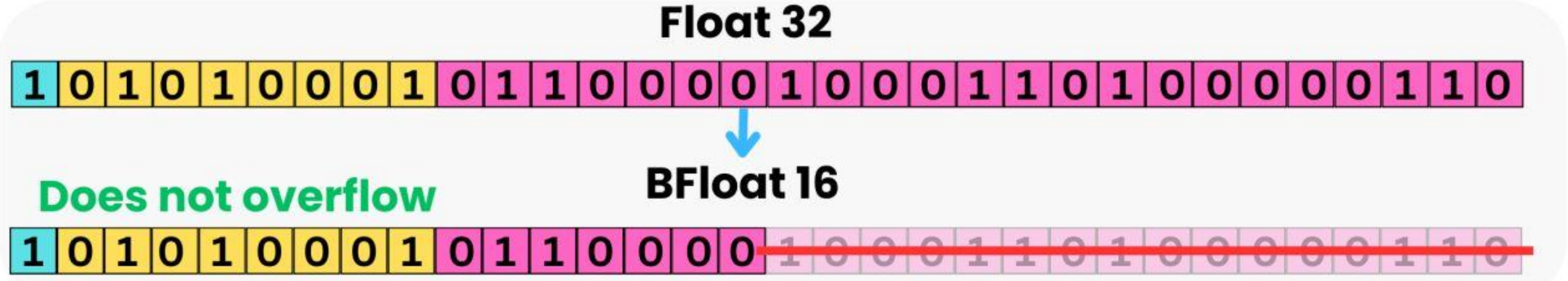
We allocate the bits differently to built different Float numbers



Converting from Float32 to Float16 can overflow



Converting from Float32 to BFloat16 is trivial



Finding Percentage Accuracy of FP32

- The smallest number: 1.17549435E-38
- The next smallest number: 1.17549449 E-38

- Find the difference between them: 1.4E-07 * E-38 . Call it Delta

- Find Delta *100/(SmallestNumber*2). This gives you percentage error, which is 5.95494E-06%, or 0.0000059549% or ~0.000006%.

Why don’t we use float/double for currency

Currency requires us being able to represent accurately and precisely up to 2 decimal places. i.e. up to 0.01 (any currency).

Using float or double, we get the following closest representations:

Float:9.999999776482582092285156 25E-3

Double:1.000000000000000020816681711 72E-2

Which is not accurate to the original decimal value 0.1. Therefore, Float and Double are not fit for the use case of currencies.