FP64, FP32, FP16, BFLOAT16, TF32, and other members of the ZOO

Grigory Sapunov · Follow

7 min read · May 17, 2020

There are many floating point formats you can hear about in the context of deep learning. Here is a summary of what are they about and where are they used.

FP80

A 80-bit IEEE 754 **extended precision** binary floating-point format typically known by the x86 implementation started from the Intel 8087 math co-processor (good old times when CPUs did not support floating point computations and FPU was a separate co-processor). In this implementation it contains:

- 1 bit sign
- 15 bits exponent
- 64 bits fraction

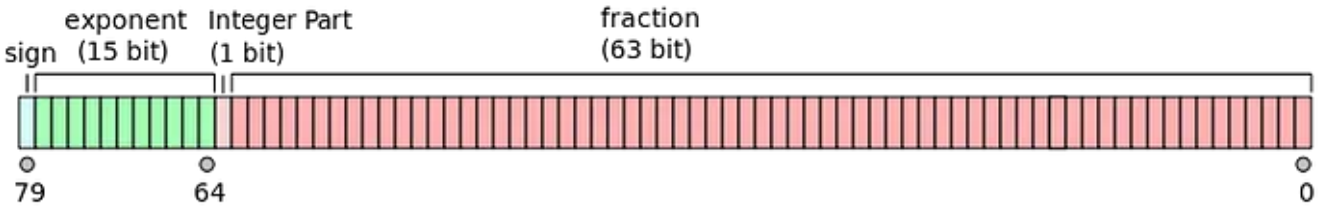


Image from Wikipedia

Range: ~3.65e−4951 to ~1.18e4932 with approximately 18 significant digits of precision.

Usage:

- The format is used for scientific computations with strong precision requirements.
- Not used in DL computations.

Software support:

- Many (but not all) C/C++ compilers implement `long double` using this 80-bit (10 bytes) format.
- Typically not supported in DL frameworks.

Hardware support:

- Natively supported by x86 CPUs (its x87 instruction subset). By default, the x87 processors all use 80-bit double-extended precision internally.
- Not supported by NVIDIA GPUs.

FP64

A 64-bit floating point, typically the IEEE 754 **double-precision** binary floating-point format with:

- 1 bit sign
- 11 bits exponent
- 52 bits fraction

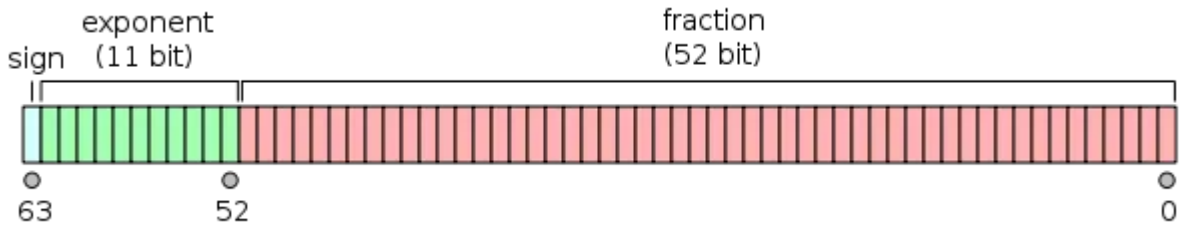


Image from Wikipedia

Range: ~2.23e-308 … ~1.80e308 with full 15–17 decimal digits precision.

Usage:

- The format is used for scientific computations with rather strong precision requirements.
- Typically not used in DL computations.

Software support:

- On most C/C++ systems represents the `double` type.
- Supported in TensorFlow (as `tf.float64`)/PyTorch (as `torch.float64` or `torch.double`).

Hardware support:

- Normally supported in x86 CPUs.
- Most GPUs, especially gaming ones including RTX series, have severely limited FP64 performance (usually 1/32 of FP32 performance instead of 1/2, see the post on

GPUs for more details).

- Among recent GPUs with unrestricted FP64 support are **GP100/102/104** in Tesla P100/P40/P4 and Quadro GP100, **GV100** in Tesla V100/Quadro GV100/Titan V and **GA100** in recently announced A100 (interestingly, the new Ampere architecture has 3rd generation tensor cores with FP64 support, the A100 Tensor Core now includes new IEEE-compliant FP64 processing that delivers 2.5x the FP64 performance of V100).

FP32

The format that was the workhorse of deep learning for a long time. Another IEEE 754 format, the **single-precision** floating-point with:

- 1 bit sign
- 8 bits exponent
- 23 bits fraction

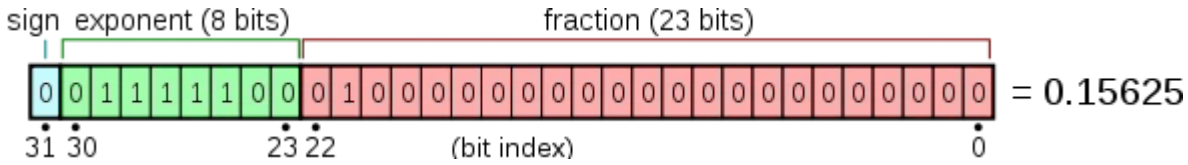


Image from Wikipedia

Range: ~1.18e-38 ... ~3.40e38 with 6–9 significant decimal digits precision.

Usage:

- The standard type for neural network computations for a long time. Weights, activations and other values in neural networks have long been represented in FP32 by default.
- For many scientific computations (especially iterative ones) the precision is not enough, leading to accumulation of errors.

Software support:

- On most C/C++ systems represents the `float` type.

- Supported in TensorFlow (as `tf.float32`)/PyTorch (as `torch.float32` or `torch.float`).

Hardware support:

- Normally supported in x86 CPUs.
- Normally supported in NVIDIA/AMD GPUs.

FP16

Again, the IEEE 754 standard format, the **half-precision** floating-point format with:

- 1 bit sign
- 5 bits exponent
- 10 bits fraction

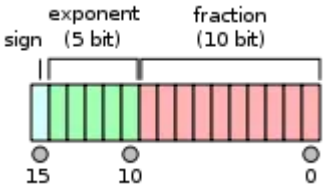


Image from Wikipedia

Range: ~5.96e−8 (6.10e−5) … 65504 with 4 significant decimal digits precision.

Usage:

- There is a trend in DL towards using FP16 instead of FP32 because lower precision calculations seem to be not critical for neural networks. Additional precision gives nothing, while being slower, takes more memory and reduces speed of communication.
- Can be used for training, typically using mixed-precision training (TensorFlow/PyTorch).
- Can be used for post-training quantization for faster inference (TensorFlow Lite). Other formats in use for post-training quantization are integer **INT8** (8-bit integer),

INT4 (4 bits) and even **INT1** (a binary value).

Software support:

- Currently not in the C/C++ standard (but there is a `short float` proposal). Some C/C++ systems support `__fp16` type. Otherwise, can be used with special libraries.
- Supported in TensorFlow (as `tf.float16`)/PyTorch (as `torch.float16` or `torch.half`).

Hardware support:

- Not supported in x86 CPUs (as a distinct type).
- Was poorly supported on older gaming GPUs (with 1/64 performance of FP32, see the post on GPUs for more details). Right now well-supported on modern GPUs, e.g. NVIDIA RTX series.

Useful links:

- Half-Precision Floating-Point, Visualized
- "Half Precision" 16-bit Floating Point Arithmetic

BFLOAT16

Another 16-bit format originally developed by Google is called "**Brain Floating Point Format**", or "bfloat16" for short. The name flows from "Google Brain", which is an artificial intelligence research group at Google where the idea for this format was conceived.

The original IEEE FP16 was not designed with deep learning applications in mind, its dynamic range is too narrow. BFLOAT16 solves this, providing dynamic range identical to that of FP32.

So, BFLOAT16 has:

- 1 bit sign
- 8 bits exponent

- 7 bits fraction

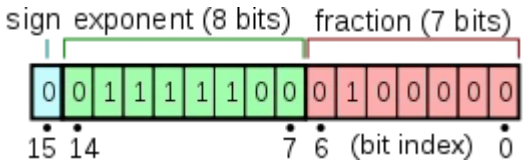
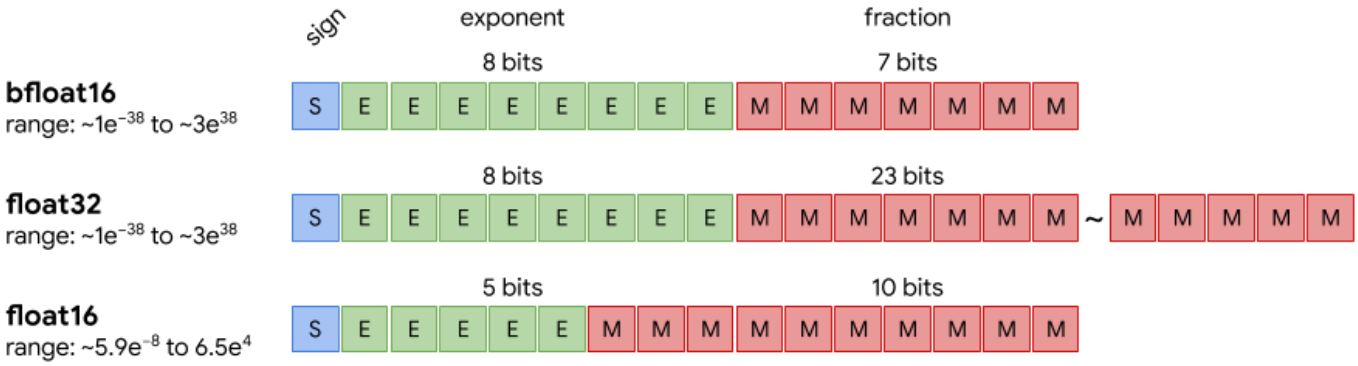


Image from WikiPedia

The bfloat16 format, being a truncated IEEE 754 FP32, allows for fast conversion to and from an IEEE 754 FP32. In conversion to the bfloat16 format, the exponent bits are preserved while the significand field can be reduced by truncation.



Source

Range: ~1.18e-38 … ~3.40e38 with 3 significant decimal digits.

Usage:

- Seems to be replacing FP16 right now. Unlike FP16, which typically requires special handling via techniques such as loss scaling, BF16 comes close to being a drop-in replacement for FP32 when training and running deep neural networks.

Software support:

- Not in the C/C++ standard. Can be used with special libraries.
- Supported in TensorFlow (as `tf.bfloat16`)/PyTorch (as `torch.bfloat16`).

Hardware support:

- CPU: Supported in modern Intel Xeon x86 (Cooper Lake microarchitecture) with AVX-512 BF16 extensions, ARMv8-A.
- GPU: Supported in NVIDIA A100 (first one to support), will be supported in future AMD GPUs.
- ASIC: Supported in Google TPU v2/v3 (not v1!), was supported in Intel Nervana NNP (now cancelled).

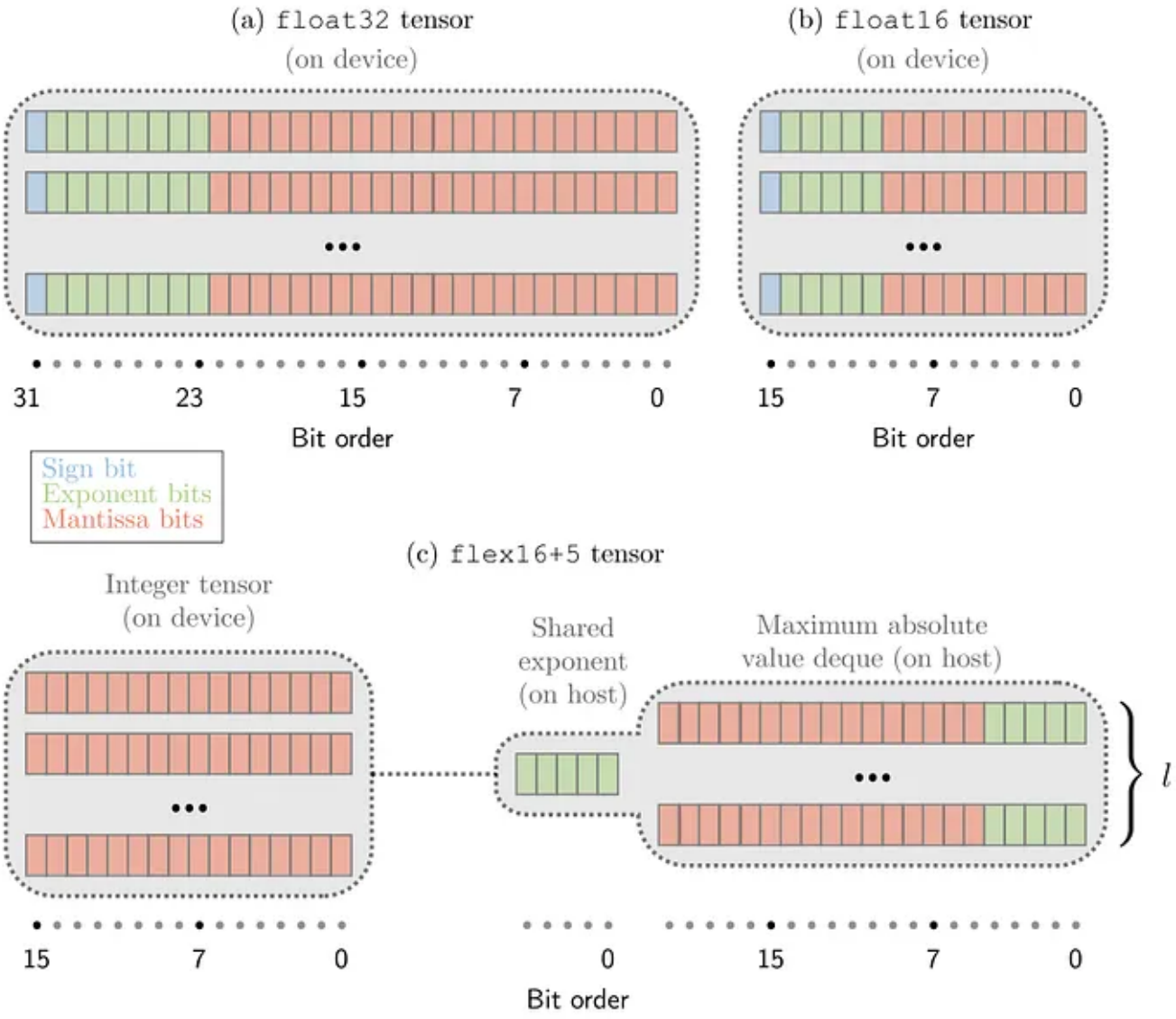
Useful links:

- Brain floating-point format (bfloat16)
- Half Precision Arithmetic: fp16 Versus bfloat16
- BFloat16: The secret to high performance on Cloud TPUs
- Using bfloat16 with TensorFlow models
- A Study of BFLOAT16 for Deep Learning Training
- bfloat16 — Hardware Numerics Definition

Flexpoint

Flexpoint is a compact number encoding format developed by Intel's Nervana used to represent standard floating point values. Later, Nervana switched to BFLOAT, then, even later, Intel cancelled Nervana processors.

Flexpoint combines the advantages of fixed point and floating point by splitting up the mantissa and the exponent part which is shared across all arithmetic execution elements. By only passing the integer value, both memory and bandwidth requirements are reduced. Additionally, this lowers hardware complexity, lowering both power and area requirements.



NIPS 2017 paper

Usage:

- Seems to be not in use.

Useful links:

- Flexpoint: An Adaptive Numerical Format for Efficient Training of Deep Neural Networks
- Flexpoint from WikiChip

TF32

TensorFloat-32, or TF32, is the new math mode in NVIDIA A100 GPUs.

TF32 uses the same 10-bit mantissa as the half-precision (FP16) math, shown to have more than sufficient margin for the precision requirements of AI workloads. And TF32 adopts the same 8-bit exponent as FP32 so it can support the same numeric range.

It is technically a 19-bit format. You can treat it as an extended-precision BFLOAT16, say “BFLOAT19" ☺ Or like reduced-precision FP32.

So, TF32 has:

- 1 bit sign
- 8 bits exponent
- 10 bits fraction

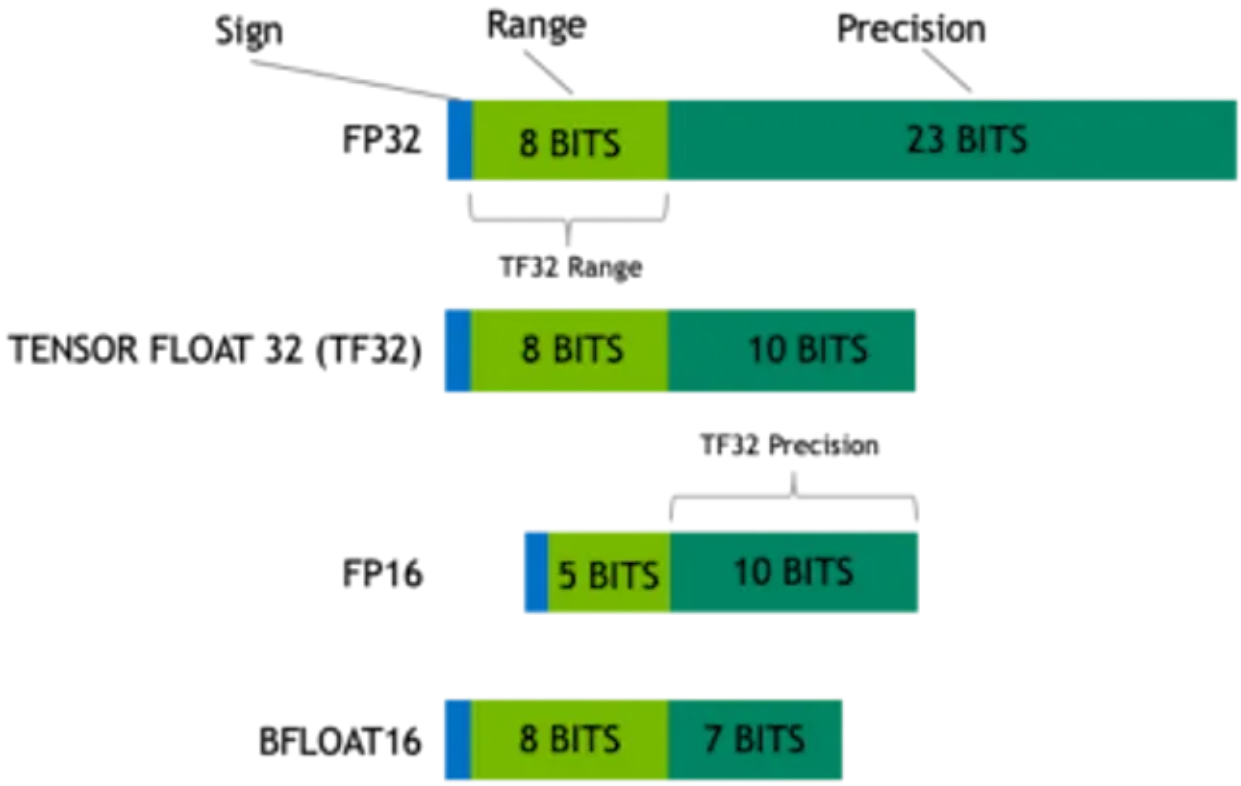


Image from NVIDIA blog post

The advantage of TF32 is that the format is the same as FP32. When computing inner products with TF32, the input operands have their mantissas rounded from 23 bits to

10 bits. The rounded operands are multiplied exactly, and accumulated in normal FP32.

TF32 Tensor Cores operate on FP32 inputs and produce results in FP32 with no code change required. Non-matrix operations continue to use FP32. This provides an easy path to accelerate FP32 input/output data in DL frameworks and HPC.

Range: ~1.18e-38 … ~3.40e38 with 4 significant decimal digits precision.

Usage:

- The big advantage of TF32 is that compiler support is required only at the deepest levels, i.e. inside the CUDA compiler. The rest of code just sees FP32 with less precision, but the same dynamic range. Exploiting TF32 will largely be a matter of tweaking callers of libraries to indicate whether TF32 is okay. TF32 exists as something that can be quickly plugged in to exploit Tensor Core speed without much work.
- Formats such as FP16 and BFLOAT16 require more work, since they involve different bit layouts. It is worth putting in efforts into using these formats, since they reduce memory bandwidth and consequently permit even faster execution. (source)

For comparison, A100's peak performances are:

- FP32 without tensor core: 19.5 TFLOPS
- TF32 tensor core: 156 TFLOPS (so, using TF32 in place of FP32 can give you an easy speed improvement).
- FP16/BF16 tensor core: 312 TFLOPS (so, thoughtfully designed switch to FP16/BF16 can give you more speed improvements, but the costs are higher).

Software support:

- Not in C/C++ standard.
- Supported in CUDA 11.

Hardware support:

- GPU: Supported in NVIDIA A100 (first one to support).

Useful links:

- TensorFloat-32 in the A100 GPU Accelerates AI Training, HPC up to 20x
- NVIDIA Ampere Architecture In-Depth

Deep Learning | Floating Point | Tf32 | Bfloat | Fp16

Follow

Written by Grigory Sapunov

1.6K Followers

ML/DL/AI expert. Software engineer with 20+ years programming experience. Loves Life Sciences. CTO and co-Founder of Intento. Google Developer Expert in ML.