WIKIPEDIA

bfloat16 floating-point format

The **bfloat16** (**Brain Floating Point**)[1][2] floating-point format is a computer number format occupying 16 bits in computer memory; it represents a wide dynamic range of numeric values by using a floating radix point. This format is a truncated (16-bit) version of the 32-bit IEEE 754 single-precision floating-point format (binary32) with the intent of accelerating machine learning and near-sensor computing.[3] It preserves the approximate dynamic range of 32-bit floating-point numbers by retaining 8 exponent bits, but supports only an 8-bit precision rather than the 24-bit significand of the binary32 format. More so than single-precision 32-bit floating-point numbers, bfloat16 numbers are unsuitable for integer calculations, but this is not their intended use. Bfloat16 is used to reduce the storage requirements and increase the calculation speed of machine learning algorithms.[4]

The bfloat16 format was developed by Google Brain, an artificial intelligence research group at Google.[5] The bfloat16 format is utilized in Intel AI processors, such as Nervana NNP-L1000, Xeon processors (AVX-512 BF16 extensions), and Intel FPGAs,[6][7][8] Google Cloud TPUs,[9][10][11] and TensorFlow.[11][12] ARMv8.6-A,[13] AMD ROCm,[14] and CUDA[15] also support the bfloat16 format. On these platforms, bfloat16 may also be used in mixed-precision arithmetic, where bfloat16 numbers may be operated on and expanded to wider data types.

Contents

bfloat16 floating-point format

bfloat16 has the following format:

- Sign bit: 1 bit
- Exponent width: 8 bits

- Significand precision: 8 bits (7 explicitly stored), as opposed to 24 bits in a classical single-precision floating-point format

The bfloat16 format, being a truncated IEEE 754 single-precision 32-bit float, allows for fast conversion to and from an IEEE 754 single-precision 32-bit float; in conversion to the bfloat16 format, the exponent bits are preserved while the significand field can be reduced by truncation (thus corresponding to round toward 0), ignoring the NaN special case. Preserving the exponent bits maintains the 32-bit float's range of ≈ 10^{-38} to ≈ 3×10^{38}.[16]

The bits are laid out as follows:

IEEE half-precision 16-bit float

sign | exponent (5 bit) | fraction (10 bit)

0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0
15	14				10	9									0

IEEE 754 single-precision 32-bit float

sign | exponent (8 bit) | fraction (23

0	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
31	30							23	22											

bfloat16

sign | exponent (8 bit) | fraction (7 bit)

0	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0
15	14							7	6						0

NVidia's TensorFloat

sign | exponent (8 bit) | fraction (10 bit)

0	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0
18	17							10	9									0

AMD's fp24 format

sign | exponent (7 bit) | fraction (16 bit)

0	0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
23	22						16	15												

Pixar's PXR24 format

sign | exponent (8 bit) | fraction (15 bit)

0	0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
23	22							15	14											

Contrast with bfloat16 and single precision

S	E	E	E	E	E	E	E	E	F	F	F	F	F	F	F	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f

Legend

- S: sign
- E: exponent
- F: fraction (trailing significand) in both formats
- f: fraction (trailing significand) in 32-bit single precision (comparative)

Exponent encoding

The bfloat16 binary floating-point exponent is encoded using an offset-binary representation, with the zero offset being 127; also known as exponent bias in the IEEE 754 standard.

- $E_{min} = 01_H - 7F_H = -126$
- $E_{max} = FE_H - 7F_H = 127$
- Exponent bias $= 7F_H = 127$

Thus, in order to get the true exponent as defined by the offset-binary representation, the offset of 127 has to be subtracted from the value of the exponent field.

The minimum and maximum values of the exponent field (00_H and FF_H) are interpreted specially, like in the IEEE 754 standard formats.

Exponent	Significand zero	Significand non-zero	Equation
00_H	zero, −0	subnormal numbers	$(-1)^{signbit} \times 2^{-126} \times$ 0.significandbits
01_H, ..., FE_H	normalized value		$(-1)^{signbit} \times 2^{exponentbits-127} \times$ 1.significandbits
FF_H	±infinity	NaN (quiet, signaling)	

The minimum positive normal value is $2^{-126} \approx 1.18 \times 10^{-38}$ and the minimum positive (subnormal) value is $2^{-126-7} = 2^{-133} \approx 9.2 \times 10^{-41}$.

Encoding of special values

Positive and negative infinity

Just as in IEEE 754, positive and negative infinity are represented with their corresponding sign bits, all 8 exponent bits set (FF_{hex}) and all significand bits zero. Explicitly,

```
val    s_exponent_signcnd
+inf = 0_11111111_0000000
-inf = 1_11111111_0000000
```

Not a Number

Just as in IEEE 754, NaN values are represented with either sign bit, all 8 exponent bits set (FF_{hex}) and not all significand bits zero. Explicitly,

```
val    s_exponent_signcnd
+NaN = 0_11111111_klmnopq
-NaN = 1_11111111_klmnopq
```

where at least one of *k, l, m, n, o, p,* or *q* is 1. As with IEEE 754, NaN values can be quiet or signaling, although there are no known uses of signaling bfloat16 NaNs as of September 2018.

Range and precision

Bfloat16 is designed to maintain the number range from the 32-bit IEEE 754 single-precision floating-point format (binary32), while reducing the precision from 24 bits to 8 bits. This means that the precision is between two and three decimal digits, and bfloat16 can represent finite values up to about 3.4×10^{38}.

Examples

These examples are given in bit *representation*, in hexadecimal and binary, of the floating-point value. This includes the sign, (biased) exponent, and significand.

```
3f80 = 0 01111111 0000000 = 1
c000 = 1 10000000 0000000 = −2
```

```
7f7f = 0 11111110 1111111 = (2⁸ − 1) × 2⁻⁷ × 2¹²⁷ ≈ 3.38953139 × 10³⁸ (max finite positive value in bfloat16 precision)
0080 = 0 00000001 0000000 = 2⁻¹²⁶ ≈ 1.175494351 × 10⁻³⁸ (min normalized positive value in bfloat16 precision and single-precision floating point)
```

The maximum positive finite value of a normal bfloat16 number is $3.38953139 \times 10^{38}$, slightly below $(2^{24} - 1) \times 2^{-23} \times 2^{127} = 3.402823466 \times 10^{38}$, the max finite positive value representable in single precision.

Zeros and infinities

```
0000 = 0 00000000 0000000 = 0
8000 = 1 00000000 0000000 = −0
```

```
7f80 = 0 11111111 0000000 = infinity
ff80 = 1 11111111 0000000 = −infinity
```

Special values

```
4049 = 0 10000000 1001001 = 3.140625 ≈ π ( pi )
3eab = 0 01111101 0101011 = 0.333984375 ≈ 1/3
```

NaNs

```
ffc1 = x 11111111 1000001 => qNaN
ff81 = x 11111111 0000001 => sNaN
```

See also

- Half-precision floating-point format: 16-bit float w/ 1-bit sign, 5-bit exponent, and 11-bit significand, as defined by IEEE 754
- ISO/IEC 10967, Language Independent Arithmetic
- Primitive data type
- Minifloat
- Google Brain

References

1. Teich, Paul (2018-05-10). "Tearing Apart Google's TPU 3.0 AI Coprocessor" (https://www.nextplatform.com/2018/05/10/tearing-apart-googles-tpu-3-0-ai-coprocessor/). *The Next Platform*. Retrieved 2020-08-11. "Google invented its own internal floating point format called “bfloat” for “brain floating point” (after Google Brain)."
2. Wang, Shibo; Kanwar, Pankaj (2019-08-23). "BFloat16: The secret to high performance on Cloud TPUs" (https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus). *Google Cloud*. Retrieved 2020-08-11. "This custom floating point format 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."
3. Tagliavini, Giuseppe; Mach, Stefan; Rossi, Davide; Marongiu, Andrea; Benin, Luca (2018). "A transprecision floating-point platform for ultra-low power computing". *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. pp. 1051–1056. arXiv:1711.10374 (https://arxiv.org/abs/1711.10374). doi:10.23919/DATE.2018.8342167 (https://doi.org/10.23919%2FDATE.2018.8342167). ISBN 978-3-9819263-0-9. S2CID 5067903 (https://api.semanticscholar.org/CorpusID:5067903).
4. Dr. Ian Cutress (2020-03-17). "Intel': Cooper lake Plans: Why is BF16 Important?" (https://www.anandtech.com/show/15631/intels-cooper-lake-plans-the-chip-that-wasnt-meant-to-exist-dies-for-you). Retrieved 2020-05-12. "The bfloat16 standard is a targeted way of representing numbers that give the range of a full 32-bit number, but in the data size of a 16-bit number, keeping the accuracy close to zero but being a bit more loose with the accuracy near the limits of the standard. The bfloat16 standard has a lot of uses inside machine learning algorithms, by offering better accuracy of values inside the algorithm while affording double the data in any given dataset (or doubling the speed in those calculation sections)."

5. Abadi, Martín; Barham, Paul; Chen, Jianmin; Chen, Zhifeng; Davis, Andy; Dean, Jeffrey; Devin, Matthieu; Ghemawat, Sanjay; Irving, Geoffrey; Isard, Michael; Kudlur, Manjunath; Levenberg, Josh; Monga, Rajat; Moore, Sherry; Murray, Derek G.; Steiner, Benoit; Tucker, Paul; Vasudevan, Vijay; Warden, Pete; Wicke, Martin; Yu, Yuan; Zheng, Xiaoqiang (2016). "TensorFlow: A System for Large-Scale Machine Learning" (https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf) (PDF). arXiv:1605.08695 (https://arxiv.org/abs/1605.08695).
6. Khari Johnson (2018-05-23). "Intel unveils Nervana Neural Net L-1000 for accelerated AI training" (https://venturebeat.com/2018/05/23/intel-unveils-nervana-neural-net-l-1000-for-accelerated-ai-training/). *VentureBeat*. Retrieved 2018-05-23. "...Intel will be extending bfloat16 support across our AI product lines, including Intel Xeon processors and Intel FPGAs."
7. Michael Feldman (2018-05-23). "Intel Lays Out New Roadmap for AI Portfolio" (https://www.top500.org/news/intel-lays-out-new-roadmap-for-ai-portfolio/). *TOP500 Supercomputer Sites*. Retrieved 2018-05-23. "Intel plans to support this format across all their AI products, including the Xeon and FPGA lines"
8. Lucian Armasu (2018-05-23). "Intel To Launch Spring Crest, Its First Neural Network Processor, In 2019" (https://www.tomshardware.com/news/intel-neural-network-processor-lake-crest,37105.html). *Tom's Hardware*. Retrieved 2018-05-23. "Intel said that the NNP-L1000 would also support bfloat16, a numerical format that's being adopted by all the ML industry players for neural networks. The company will also support bfloat16 in its FPGAs, Xeons, and other ML products. The Nervana NNP-L1000 is scheduled for release in 2019."
9. "Available TensorFlow Ops | Cloud TPU | Google Cloud" (https://cloud.google.com/tpu/docs/tensorflow-ops). *Google Cloud*. Retrieved 2018-05-23. "This page lists the TensorFlow Python APIs and graph operators available on Cloud TPU."
10. Elmar Haußmann (2018-04-26). "Comparing Google's TPUv2 against Nvidia's V100 on ResNet-50" (https://web.archive.org/web/20180426200043/https://blog.riseml.com/comparing-google-tpuv2-against-nvidia-v100-on-resnet-50-c2bbb6a51e5e). *RiseML Blog*. Archived from the original (https://blog.riseml.com/comparing-google-tpuv2-against-nvidia-v100-on-resnet-50-c2bbb6a51e5e) on 2018-04-26. Retrieved 2018-05-23. "For the Cloud TPU, Google recommended we use the bfloat16 implementation from the official TPU repository with TensorFlow 1.7.0. Both the TPU and GPU implementations make use of mixed-precision computation on the respective architecture and store most tensors with half-precision."
11. Tensorflow Authors (2018-07-23). "ResNet-50 using BFloat16 on TPU" (https://github.com/tensorflow/tpu/tree/0ece10f6f4e523eab79aba0247b513fe57d38ae6/models/experimental/resnet_bfloat16). *Google*. Retrieved 2018-11-06.
12. Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, Rif A. Saurous (2017-11-28). TensorFlow Distributions (Report). arXiv:1711.10604 (https://arxiv.org/abs/1711.10604). Bibcode:2017arXiv171110604D (https://ui.adsabs.harvard.edu/abs/2017arXiv171110604D). Accessed 2018-05-23. "All operations in TensorFlow Distributions are numerically stable across half, single, and double floating-point precisions (as TensorFlow dtypes: tf.bfloat16 (truncated floating point), tf.float16, tf.float32, tf.float64). Class constructors have a validate_args flag for numerical asserts"
13. "BFloat16 extensions for Armv8-A" (https://community.arm.com/developer/ip-products/processors/b/ml-ip-blog/posts/bfloat16-processing-for-neural-networks-on-armv8_2d00_a). *community.arm.com*. Retrieved 2019-08-30.

14. "ROCm version history" (https://github.com/RadeonOpenCompute/ROCm/blob/8bd9a527405cb466d45b3b343a33434e79b1d387/version_history.md#miopen-20). *github.com*. Retrieved 2019-10-23.
15. "CUDA Library bloat16 Intrinsics" (https://docs.nvidia.com/cuda/cuda-math-api/group__CUDA__MATH__INTRINSIC__BFLOAT16.html#group__CUDA__MATH__INTRINSIC__BFLOAT16).
16. "Livestream Day 1: Stage 8 (Google I/O '18) - YouTube" (https://www.youtube.com/watch?v=vm67WcLzfvc&t=2555). *Google*. 2018-05-08. Retrieved 2018-05-23. "In many models this is a drop-in replacement for float-32"