About 8-bit floating-point numbers for machine learning: the IEEE 3109 project

Nathalie Revol
INRIA - ENS de Lyon

Bordeaux, 18-06-2024

Introduction

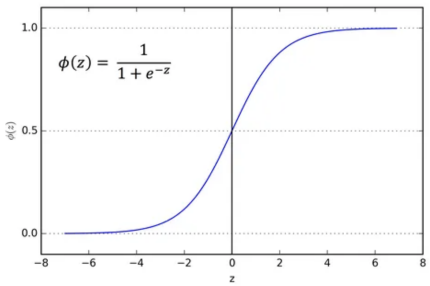
Artificial Intelligence, Machine Learning

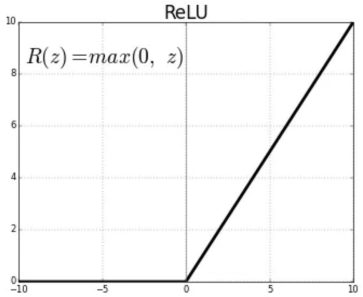
Nowadays, frequently:

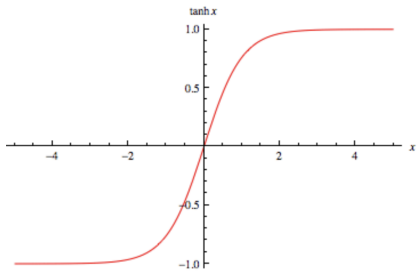
- neural network,
- neurons arranged in layers and computations on each "neuron",
- communication from one layer to the next

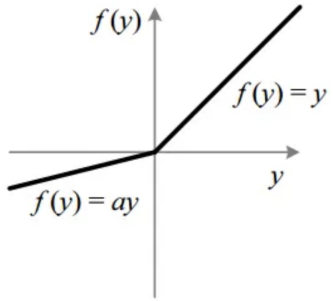
Typical computation on a neuron:

$f(w.a+b)$ where f is one of the nonlinear functions depicted below, w is a vector of weights, a is the vector of inputs and b is a bias.









Training:

- forward = evaluation as in the previous slide
- backward to adjust the weights = use of the gradient

Inference = use:

- only forward,
- only evaluation,
- no need to be accurate,
- range/dynamic is known

With the current expansion of artificial intelligence,
what is wanted: reduction in terms of

- silicon area,
- communication time,
- computation and thus energy consumption

Idea: use (floating-point) numbers stored on less bits.
Existing solutions: FP8, OCP, 6-bit, 4-bit...

In this talk: introduction to the FP8 format as defined by IEEE 3109.

THE IEEE 3109 working group

- creation end 2022
- officers: Kiran Gunnam, Leonard Tsai, Jeffrey Sarnoff
- co-editors: Andrew Fitzgibbon, Guy Lemieux, Jeffrey Sarnoff
- 118 registered persons, 35 voting persons
- meeting every other week, Monday 18h-19h30

Agenda

Formats

Special values

Rounding modes

Coming soon

Conclusion and questions

Digression number 1: RNTO

Digression number 2: RSTO

Agenda

FP8 formats

FP8 formats: `binary8p`p where $p \in \{1, 2, \ldots 7\}$.

The different fields of x: sign, exponent, significand :

sign	1 bit	0 if $x \geq 0$, 1 otherwise
exponent	$8 - p$ bits	values symmetric around 0
significand	$p - 1$ bits	p bits, including the hidden bit

FP8 formats

FP8 formats: `binary8p`p where $p \in \{1, 2, \ldots 7\}$.

The different fields of x: sign, exponent, significand :

sign	1 bit	0 if $x \geq 0$, 1 otherwise
exponent	$8 - p$ bits	values symmetric around 0
significand	$p - 1$ bits	p bits, including the hidden bit

Discussion: Why use one bit for the sign? What about unsigned floating-point numbers?

FP8 formats: summary

(copy of Table 2 of the interim draft report of Sep. 2023 - Feb. 2024)

Symbol	Parameter Description	Derived Value	binary8pP 7	6	5	4	3	2	1	IEEE754-2019 16	32	64
K	storage (bits)	K	8	8	8	8	8	8	8	16	32	64
P	precision (bits)	P	7	6	5	4	3	2	1	11	24	53
S	sign (bits)	1	1	1	1	1	1	1	1	1	1	1
W	exponent (bits)	K - P	1	2	3	4	5	6	7	5	8	11
T	trailing significand (bits)	P - 1	6	5	4	3	2	1	0	10	23	52
SE	all-special exponent	SE	0	0	0	0	0	0	1	1	1	1
emax	maximum exponent	$2^{W-\mathbf{1}} - 1$	0	1	3	7	15	31	63	15	127	1023
emin	minimum exponent	SE - emax	0	-1	-3	-7	-15	-31	-62	-14	-126	-1022
bias	exponent bias	1 - emin	1	2	4	8	16	32	63	15	127	1023

Agenda

Special values

- NaN
- 0
- infinities
- subnormals
- extremal values

Bonus in this talk: a comparison with IEEE 754.

Special value: NaN

Binary8 value sets shall include exactly one NaN, encoded as $0x80 = 10000000$, which shall not signal.

Rationale: with 8 bits, there are only 256 possible values. Reserving 2^p values to be NaN – as in IEEE 754-2019 – has been considered as a waste of a too scarce resource.

Special value: 0

Binary8 formats shall have exactly one zero, encoded as $0x00 = 00000000$. This zero value is nonnegative.

Rationale: the inclusion of negative zero (represented as $0x80 = 10000000$) would incur the cost of an additional code point. Again, the number of stored values is too small to allow for two representations of 0.

Instead, the representation of "-0" is used for NaN.

Given the decision to encode only a single NaN, placing that NaN at the negative zero code point enables the strictly positive and strictly negative number ranges to be symmetric.

Special values: infinities

Binary8 formats shall include positive and negative infinities, encoded as $0x7f = 01111111$ for $+\infty$ and $0xff = 1111111$ for $-\infty$, respectively.

Difficulty: $1/1/-\infty = 1/0 = +\infty$???
Rather $1/0 =$ NaN.

Discussion (in May 2024) between infinities and saturation, both may be available.
Saturation needs a definition (before or after rounding? does it make a difference?)

This decision causes a reduction in dynamic range (252 values rather than 254), while offering improved numerical robustness in important machine learning use cases.
Examples of such usage are:

- Mask values, for example, in Transformer models in machine learning.
- Representation of overflow, for example, to adjust dynamic loss scaling factors.

Special values: subnormals

Binary8 value sets shall include subnormals.

Normal value: when at least one bit of the representation of the exponent E is 1,
then the significand $x_1 \dots x_n$ is interpreted as $1.x_1 \dots x_n \times 2^{E-\text{bias}}$.

Subnormal: when all bits of the exponent's representation are 0,
then the significand $x_1 \dots x_n$ is interpreted as $0.x_1 \dots x_n \times 2^{1-\text{bias}}$.

Subnormal numbers extend the dynamic range of floating-point values and induce equal quantization steps close to zero. They can be useful when training models, where it is common to represent near-zero values for gradients. Subnormals can also be useful to represent random values pulled from certain distributions. For example, model weights are initialized to small random values at training. Subnormals are uniformly spaced around zero, and values near zero are more probable in Gaussian-like distributions values. Finally, formats with narrow exponent widths necessarily have a limited range; subnormals extend this range by a power of 2 for every bit in the trailing significand

Special values: extremal values

(copy of Table 3 of the interim draft report of Sep. 2023 - Feb. 2024)

Format	minSubnormal	maxSubnormal	minNormal	maxNormal
binary8p1	N/A	N/A	1×2^{-62}	1×2^{63}
binary8p2	1×2^{-32}	1×2^{-32}	1×2^{-31}	1×2^{31}
binary8p3	1×2^{-17}	$3/2 \times 2^{-16}$	1×2^{-15}	$3/2 \times 2^{15}$
binary8p4	1×2^{-10}	$7/4 \times 2^{-8}$	1×2^{-7}	$7/4 \times 2^{7}$
binary8p5	1×2^{-7}	$15/8 \times 2^{-4}$	1×2^{-3}	$15/8 \times 2^{3}$
binary8p6	1×2^{-6}	$31/16 \times 2^{-2}$	1×2^{-1}	$31/16 \times 2^{1}$
binary8p7	1×2^{-6}	$63/32 \times 2^{-1}$	1×2^{0}	$63/32 \times 2^{0}$

Agenda

Rounding modes

- `roundTowardPositive`
- `roundTowardNegative`
- `roundTowardZero`
- `roundToNearestTiesToEven`
- `roundToNearestTiesToAway`
- `roundToNearestTiesToZero`

Rounding modes

- roundTowardPositive
- roundTowardNegative
- roundTowardZero
- roundToNearestTiesToEven
- roundToNearestTiesToAway
- roundToNearestTiesToZero
- roundTiesToOdd
- stochastic rounding

Agenda

Operations

Current discussion: `abs`, `neg` have been mandated by IEEE 3109.
What about `recip` $= 1/x$?
What about other operations?

Rounding modes: double rounding?

Early June's discussion about the reference computation: exact mathematics or `binary32`?

Risk with binary32: double rounding
Feature of FP8: it is possible to do exhaustive tests (only 256 values) to detect whether double rounding occurs.

Conclusion: stick to the result computed exactly and then rounded?

Operations

Considered operations are those related to machine learning applications.

What about more general ones, will there be a libm? For instance `tanh` and `atanh` are used as activation functions, will they be defined?

Agenda

Conclusion

This is an ongoing work and all of the final specifications of the future standard are not yet chosen.

Questions

- Will it be usable for interval arithmetic? Namely, will **directed rounding modes** be available?
- Is the machine learning community interested in interval arithmetic, for instance to guarantee some results?
- Will it be used by applications outside machine learning?
- If it is used by applications such as robotics or control theory, how will it impact computations and results?

Credits and References

Figures taken from:

- activation function: `https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6`
- stochastic rounding: Croci M, Fasi M, Higham NJ, Mary T, Mikaitis M.: *Stochastic rounding: implementation, error analysis and applications.* R. Soc. Open Sci. 9: 211631, 2022. `https://doi.org/10.1098/rsos.211631`

Credits and References

Other useful references:

- web page of the IEEE 3109 working group:
  `https://sagroups.ieee.org/p3109wgpublic/`
- interim draft report of IEEE 3109:
  `https://github.com/P3109/Public/blob/main/SharedReports/P3109WGInterimReport.pdf`
- OCP 8-bit Floating Point Specification (OFP8):
  `https://www.opencompute.org/documents/ocp-8-bit-floating-point-specification-ofp8-revision-1-`
- IEEE 754 standard:
  `https://ieeexplore.ieee.org/document/8766229`

Agenda

Rounding to Nearest Ties to Odd

Definition: this is a round-to-nearest mode. When there is a tie, that is, when the number to be rounded is exactly the middle of two representable numbers, it is rounded to the one whose significand ends with a "1" (that is, it is odd).

Why is it useful? to perform double rounding without trouble (cf. Boldo & Melquiond).

Rounding to Nearest Ties to Odd

Why is it useful? to avoid double rounding's issues.

Example using binary8p3:
$x = 3/1024$, $y = 49152/1$, $z = 1/131072 = 2^{-17}$
using `binary64s`, $\texttt{fma}(x, y, z) = x \times y + z = 144.000007629453$
using `binary32s`, $\widehat{\texttt{fma}}(x, y, z) = 144.0$
down-converting, the best fit in `binary8p3`

converting from `binary64s`: 160.0
converting from `binary32s`: 128.0
144 is exactly halfway between 128 and 160
using `RoundNearestToEven` with 144.0 gives 128.0
using `RoundNearestToOdd` with 144.0 gives 160.0

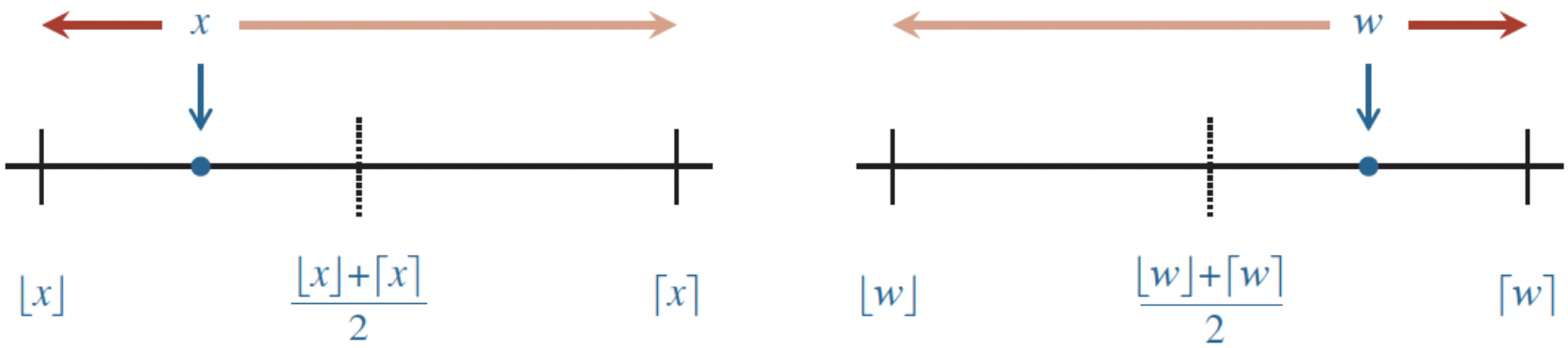
the exact result is closer to 160.0.

This is just an example, but the general case is true.

Agenda

Stochastic Rounding (1/4)

Stochastic rounding: when $x \in \mathbb{R}$ lies between two floating-point numbers $\underline{x}$ and $\bar{x}$, it is rounded to $\underline{x}$ with probability $1 - q$ and to $\bar{x}$ with probability q.



Frequently, q is $\frac{x-\underline{x}}{\bar{x}-\underline{x}}$.

Stochastic Rounding (2/4)

Advantages:

1. roundoff error grows as $\sqrt{n}u$ where n is the number of operations and u is the machine epsilon, for many algorithms (e.g. dot product),
2. no, or less, stagnation: e.g. $1.0 + u/4 + u/4 + \ldots + u/4$ returns 1.0 with rounding-to-nearest and a result closer to the exact result with stochastic rounding;
   in other words, adding tiny values (for the considered precision) still yields a correct answer.

Stochastic Rounding (3/4)

Already frequently considered for machine learning, to update the weights (during training) and avoid stagnation.

Already

- implemented by GraphCore (IPU),
- patented: IBM, NVidia, ARM,
- included in the Intel Loihi and the SpiNNaker2 digital neuromorphic processors.

Stochastic Rounding (4/4)

Interval computations make heavy use of `roundTowardPositive` and `roundTowardNegative` infinities.

How can our community take advantage of this rounding mode?

Is it possible to define a hybrid between interval arithmetic / "stochastic" arithmetic / ... using this rounding mode and getting sets "likely to contain the results"?