# Range Extension with Supernormals for Mixed-Precision 8-bit DNN Training

Shing Wai Pun
*UBC Vancouver, Canada*
*shingwaipun@gmail.com*

Bozhang Bao
*UBC Vancouver, Canada*
*baobozhangbbz@gmail.com*

Silviu-Ioan Filip
*INRIA Rennes, France*
*silviu.filip@inria.fr*

Guy Lemieux
*UBC Vancouver, Canada*
*lemieux@ece.ubc.ca*

John V. Kim
*Univ. Toronto, Canada*
*johnvkimec@gmail.com*

Nazar Misyats
*ENS Rennes, France*
*nazar.misyats@gmail.com*

Nirvik Pande
*Austin, USA*
*nirvikpande@gmail.com*

Victor Ravain
*Centrale Nantes, France*
*victorravain@hotmail.fr*

Robert Sherrick
*TTU, USA*
*rsherric@ttu.edu*

***Abstract*—To date, the most promising methods for 8-bit DNN training use two different floating-point formats: a 5-bit exponent for greater range on gradients in the backwards pass, and a 4-bit exponent with less range but greater precision for representing weights and activations in the forward pass. Ordinarily, subnormal representations are also used to extend the range and provide gradual underflow, but subnormals require considerable special-case logic which increases the cost of area-critical operators such as multiply-accumulate units. Instead, this paper proposes to replace traditional subnormals with what we call supernormal values where mantissa bits are converted into exponent bits at both the subnormal/zero and infinity ends of the range. Hence, supernormals provide maximal range extension by sacrificing some precision at both ends. The logic required to implement supernormals is a trivial change to existing logic that processes normalised values, thus avoiding the logic overhead of subnormals while providing a larger range extension. We use the range extension provided by supernormals in new mixed-precision 8-bit formats to showcase training results that are comparable to the state-of-the-art while showing an area savings.**

## 1. Introduction

The current trend in boosting the efficiency of DNN training accelerators is to use ever-smaller smaller floating-point formats as much as possible for computing and storage. Smaller element sizes facilitate higher performance by reducing memory footprint, bandwidth demand, power consumption, and cost per operation.

In particular, 8-bit floating-point (FP8) formats have become attractive in recent years, partly due to the availability of dedicated hardware for FP8 arithmetic [1] and various standardization efforts such as the OpenCompute Project FP8 specification [2] and the IEEE P3109 Working Group's Interim Report [3]. Going back to 2019 [4], a growing consensus among such studies are recommendations for two different formats to be used in 8-bit DNN training: (a) for gradients, use 5 bits of exponent for a sufficiently large representation range, and (b) for weights and activations, use 3 bits of mantissa to ensure adequate numerical accuracy. These formats are commonly called E5M2 and E4M3 based on the number of exponent and mantissa bits.[1]

The dynamic range of values used during training of ResNet-32 on CIFAR-100 varies for weights (roughly 25 binades, from $2^{-25}$ to $2^{0}$), activations (roughly 30 binades, from $2^{-25}$ to $2^{5}$) and gradients (roughly 35 binades, from $2^{-40}$ to $2^{-5}$) [5]. We have reproduced our own training histograms with ResNet-20 in Figure 1. The range required by gradients exceeds the natural range of E5M2 (31 binades without subnormals, 33 binades with subnormals). A property shared by all distributions is a lopsided histogram, where the most-frequently used values crowd the upper end of the range (towards infinity), leaving a long tail at the lower end (towards zero). To closely match this usage, the OCP FP8 formats and the preliminary IEEE P3109 8-bit formats all include subnormal representations, thereby broadening the range at the small end where it is in demand.

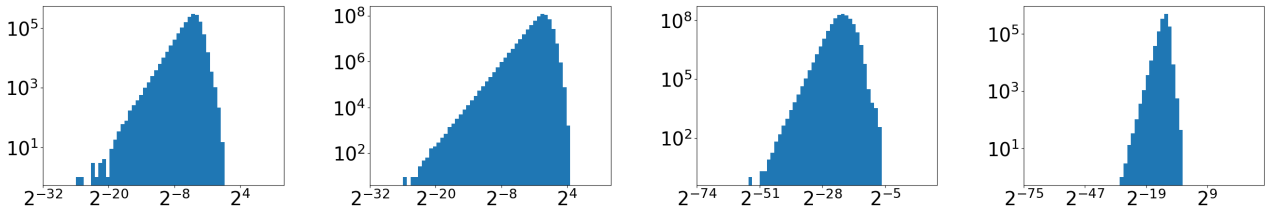
However, handling subnormals requires considerable special-case logic, which can have a big impact on the cost of area-critical operators such as multiply-accumulate (MAC) units. In the multiplier input stage, subnormals typically require renormalizing (which includes a count-leading-zero, variable-left-shift and an exponent-adjust for each operand) prior to entering the multiplier array. This subnormal overhead can be significant in small precision formats; removing subnormal inputs can save 25% to 40% of the multiplier logic [6] in an 8-bit fused MAC design. To save area, some commercial cores flush subnormals to zero.

The objectives of this paper are to simplify subnormal representations while increasing dynamic range. Ideally, this would also eliminate the need to use two different numerical formats (E5M2 and E4M3) during 8-bit DNN training. Although NVIDIA has shown both formats can be supported together in hardware, there is some area cost. However, there is considerably more complexity added to software frameworks because both the training algorithms and the DNN models need to select the best data types. This extra complexity does not exist with 16-bit or larger formats.

To meet these objectives, this paper presents an al-

1. In this paper, we precisely use the current IEEE P3109 encodings [3] named `binary8p3` and `binary8p4`, corresponding to E5M2 and E4M3, respectively. P3109 and OCP FP8 formats use a different exponent bias.

2576-2265/25/$31.00 ©2025 IEEE
DOI 10.1109/ARITH64983.2025.00013



(a) Weights (b) Activations (c) Gradients wrt Activations (d) Gradients wrt Weights

Figure 1: ResNet-20 training distributions (`binary32` without DLS; x-axis are tensor values, y-axis is count).

ternative to subnormal encoding called *supernormals*. It does this by replacing the subnormal encoding bin with normalized encodings and converting all mantissa bits into an exponent offset. This range extension can also be done at the upper end by replacing the NaN-encoding bin with supernormals. This provides power-of-2 spacing of values at both ends. While there is some loss of precision, it is limited to the extreme ends of range and produces a more gradual underflow/overflow. Since all values are normalized, no logic is needed to renormalize subnormal mantissas. In addition, supernormals offer range expansion not present in OCP, P3109, or posits [7]: the number of binades allocated to range expansion is a design parameter.

The rest of this paper explores the use of supernormals in 8-bit DNN training to show its viability as a new format.

## 2. Background

### 2.1. Mixed-Precision Training

The training acceleration landscape is dominated by mixed-precision training (MPT) approaches [5], [8]–[12]. Master copies of weights are generally kept and updated in high precision. Whereas `binary32` single-precision arithmetic (E8M23) was once common [8], researchers now use 16-bit encodings [9]–[11] such as `binary16` half-precision (E5M10) and `bfloat16` (E8M7). A reduced memory footprint enables larger models, lower power, and faster training. Higher-precision values are then cast down to a lower precision, e.g. to 8-bit [9], for use in forward and backward GEMM computations. While the GEMM outputs are represented with a higher precision, all inputs (weights, activations, and gradients) are encoded with low precision. The low-precision gradients are, in the end, used to update high-precision master copies of the weights.

### 2.2. Scaling Strategies

The danger with using low-precision formats, especially during training, is that they have a much reduced dynamic range compared to a 32-bit baseline. This tends to be problematic, especially when computing gradients.

A standard approach [8], [9], [11] is to globally scale the loss function (by a value typically ranging from $2^8$ to $2^{20}$) before executing backpropagation if a NaN or infinity appears in previously computed tensors, and rescale the computed gradients back at the end before performing the parameter updates. Used proactively, this dynamic loss scaling (DLS) strategy shifts quantities that would normally underflow or overflow back into a representable range.

This approach is too coarse-grained and insufficient in certain contexts [11]. In such cases, DLS can be complemented or replaced with tensor scaling (TS) approaches [11] or fine-grained sub-tensor (*i.e.,* block) methods [13], [14].

TABLE 1: 8-bit Encodings and Values (sign bit omitted)

| Pfx/Exp/Mant | Signif. | Exponent | Numeric Value |
|---|---|---|---|
| **P3109 `binary8p3` (E5M2)** | | | |
| `1111111//` | - | - | $\infty$ |
| `/eeeee/mm` | 1.mm | eeeee–16 | $\{2^{-15}$ to $2^{15}\} \times (1 + mm/4)$ |
| `00000//mm` | 0.mm | –15 | $2^{-15} \times mm/4$ |
| `0000000//` | 0.0 | - | 0 (*NaN* when negative) |
| * eeeee ranges from 00001 to 11111 | | | |
| **Proposed Supernormal, 1 binade (E5M2B1)** | | | |
| `1111111//` | - | - | $\infty$ |
| `11111/EE`$^U$`/` | 1.0 | EE+15 | $2^{15}, 2^{16}, 2^{17}$ |
| `/eeeee/mm` | 1.mm | eeeee–16 | $\{2^{-15}$ to $2^{14}\} \times (1 + mm/4)$ |
| `00000/EE`$^L$`/` | 1.0 | EE–19 | $2^{-18}, 2^{-17}, 2^{-16}$ |
| `0000000//` | 0.0 | - | 0 (*NaN* when negative) |
| * eeeee ranges from 00001 to 11110, EE$^U \neq 11$, EE$^L \neq 00$ | | | |
| **Proposed Supernormal, 2 binades (E5M2B2)** | | | |
| `1111111//` | - | - | $\infty$ |
| `1111/EEE`$^U$`/` | 1.0 | EEE+14 | $2^{14}$ to $2^{20}$ |
| `/eeeee/mm` | 1.mm | eeeee–16 | $\{2^{-14}$ to $2^{13}\} \times (1 + mm/4)$ |
| `0000/EEE`$^L$`/` | 1.0 | EEE–22 | $2^{-21}$ to $2^{-15}$ |
| `0000000//` | 0.0 | - | 0 (*NaN* when negative) |
| * eeeee ranges from 00010 to 11101, EEE$^U \neq 111$, EEE$^L \neq 000$ | | | |

## 3. Supernormal Floating-Point Representation

We introduce a novel 8-bit floating-point format incorporating a supernormal regime, designed to extend the representable numerical range for very large and very small values. In conventional floating-point systems, the largest exponent value is typically reserved for NaNs and infinities, and the smallest exponent value represents subnormals. In our proposed format, shown in Table 1, these largest and smallest exponent encodings instead indicate that the mantissa field encodes an additional offset to the exponent, enabling representation of much larger or smaller values. The number of largest and smallest exponent encodings

TABLE 2: Fused MAC Unit Area and Breakdown (# ALMs)

| | Mult. Inputs | Mult. | Adder (E5M10) | MAC |
|---|---|---|---|---|
| (a) | E5M3 | 34.0 | 188.0 | 222.5 |
| (b) | E5M3-nosub | 20.0 | 188.0 | 208.5 |
| (c) | E5M2 | 27.0 | 182.5 | 210.0 |
| (d) | E5M2B1 | 24.5 | 185.0 | 210.0 |

that are converted to the supernormal regime is specified in `binades` with a design parameter $B$. The use of supernormals provides a range extension at large and small values, sacrificing precision in the supernormal range.

The remaining representable values within the normalized range use the IEEE P3109 8-bit encoding rules.

Supernormals aim to better represent the typical value ranges observed during DNN training, where extremely large or small values are needed but less commonly encountered [5]. The ability to capture these extreme values is important for convergence, especially at low precision.

## 4. Experimental Results

Table 2 reports the number of Intel FPGA ALMs for different *fused* MAC unit designs. Fused means multiplier outputs are not rounded or normalized before the accumulator. In all cases, the accumulator uses a `binary16` (E5M10) adder including subnormals, and no DSP blocks are needed due to narrow multiplier inputs. The 9-bit MAC unit in row (a) supports both E5M2 and E4M3 inputs. Rows (b) and (c) save logic area in traditional ways. Row (b) 'nosub' replaces subnormal code points on the multiplier inputs with normalized values, reducing dynamic range. In contrast, row (c) uses an 8-bit MAC (E5M2) but requires similar area. Finally, row (d) uses E5M2B1, the proposed new format where supernormals replace one binade at each end of the range to save area and expand dynamic range by $2^3$.

We use two models to test the training flexibility of our proposed encodings. The first is a 0.27M parameter ResNet-20 convolutional neural network architecture [15] on the CIFAR10 image recognition dataset. The second is a 15M parameter version of the Llama 2 [16] family of large language models (LLMs) trained for text generation on the TinyStories [17] dataset. The training hyperparameter setups are the same previous work for ResNet [15] and the Llama2 repository[2]. All tests are performed on an NVIDIA 4090 RTX GPU with 24GB of RAM.

The MPT approach is identical to previous work [5], [11], as discussed in Sec. 2.1. Master copies of the weights are kept in high precision, either a 32-bit or a 16-bit format. In contrast, all the input signals to GEMM computations during the forward and backward pass are downcast to an 8-bit representation. In general, some form of dynamic loss scaling and/or tensor/block scaling is needed to account for the loss in range when using such narrow formats.

2. See https://github.com/karpathy/llama2.c

To achieve this, we extended `mptorch`[3], a PyTorch-based framework for custom-precision arithmetic. Our formats use a quantization function applied before multiplication in GEMM operations, emulating the usage of a mixed-precision MAC. Denoting the quantization function by $Q$ and using the forward pass of a linear layer as an example, we have $\mathbf{z} = Q(\mathbf{W}^T)Q(\mathbf{x}) + \mathbf{b}$, where $\mathbf{W}$ is the weight matrix, $\mathbf{x}$ is the input activation matrix, and $\mathbf{z}$ is the output activation matrix. The intermediate accumulations are not quantized and are represented in either IEEE 754 `binary16` (E5M10) or `binary32` (E8M23) formats.

Results are summarized in Table 3. The dynamic loss scaling (DLS) [8] and tensor scaling (TS) [11] columns indicate whether DLS or TS is required for convergence during MPT.[4] The first row for each networks gives the baseline training results using IEEE-754 `binary32` arithmetic and serves as a comparison point for all other mixed-precision configurations.

The second group of rows (R2/L2 to R4/L4) compare using E5M2 and E4M3 8-bit encodings. It is common practice in most 8-bit MPT configurations to use 4-bit exponents for weights/activations and 5-bit exponents for gradients. Rows R2/L2 and R3/L3 use the P3109 encodings group for `binary8p4` (E4M3) and `binary8p3` (E5M2) formats, which include subnormal values. The dynamic ranges for these two formats correspond to approximately 18 and 33 binades, respectively. The following row (R4) removes subnormals from these formats by treating the smallest binade as part of the normalized range, leading to 31 binades of dynamic range for E5M2-nosub. While this makes logic simpler, and is effective for the ResNet example, the limited dynamic range is not sufficient to ensure convergence for the LLM example.

Instead, the proposed supernormal representation from Sec. 3, E5M2B1, improves range to 36 binades with training results shown in rows R5/L5. Varying the number of supernormal binades showed that a one-binade configuration offered the best results when DLS and/or TS were used.

It is promising to note that the parameterizable range extension enabled by the number of binades may in certain cases simplify training logic and might remove the need for scaling approaches. This is the case for the ResNet-20 training example, where the 4-binade supernormal encodings (E5M2B4) in row R6 reach baseline accuracy levels. This is not the case for the Llama2 (not shown). Nevertheless, this suggests that supernormal-based encodings could be more easily used in edge training scenarios where we expect models to be relatively small.

## 5. Conclusions

This paper has proposed a replacement for subnormals, called supernormals. Dynamic range can be extended by replacing mantissa bits at both upper and lower ends of the

3. See https://github.com/mptorch/mptorch

4. TS is not available in our ResNet-20 model.

TABLE 3: Accuracy results for a ResNet-20 CNN and a Llama2 LLM in a mixed-precision training scenario.

| | | Weights/Activations | Gradients | GEMM Accumulator | DLS [8] | TS [11] | Val. Acc./Loss |
|---|---|---|---|---|---|---|---|
| ResNet-20 | (R1) | E8M23 | E8M23 | E8M23 | no | N/A | 91.72% |
| | (R2) | E4M3 | E5M2 | E8M23 | req. | N/A | 91.85% |
| | (R3) | E5M2 | E5M2 | E5M10 | req. | N/A | 91.31% |
| | (R4) | E5M2-nosub | E5M2-nosub | E5M10 | req. | N/A | 91.34% |
| | (R5) | E5M2B1 | E5M2B1 | E5M10 | req. | N/A | 91.61% |
| | (R6) | E5M2B4 | E5M2B4 | E5M10 | no | N/A | 91.46% |
| llama2.c | (L1) | E8M23 | E8M23 | E8M23 | no | no | 1.14736 |
| | (L2) | E4M3 | E5M2 | E8M23 | req. | req. | 1.14833 |
| | (L3) | E5M2 | E5M2 | E5M10 | req. | req. | 1.16494 |
| | (L4) | E5M2-nosub | E5M2-nosub | E5M10 | req. | req. | diverges |
| | (L5) | E5M2B1 | E5M2B1 | E5M10 | req. | req. | 1.16629 |

Notes: E8M23 and E5M10 are the IEEE 754 formats, `binary32` and `binary16`, respectively. E5M2 and E4M3 are the IEEE P3109 formats, `binary8p3` and `binary8p4`, respectively. E5M2-nosub replaces subnormals with normalized values. E5M2B$b$ is `binary8p3` with $2b$ total binades of supernormals.

represented range with an additional exponent offset, resulting in a 1-bit mantissa in those regions. Additional range extension at either end is possible by replacing multiple binades at either end with supernormals. Since all values are normalized, values can be manipulated without adding subnormal complexities. Our area results demonstrate that one-binade supernormals (E5M2B1) are more area efficient than formats with subnormals and rival formats without subnormals. Furthermore, our training results show that supernormals on ResNet-20 can achieve comparable accuracy, and with enough binades one single numeric format can eliminate the need to perform dynamic loss scaling. On Llama2, training could not converge without subnormals, but it did converge with supernormals at a validation loss within 1–2% of the `binary32` baseline. This demonstrates that supernormals require less area, are nearly competitive with `binary32`, and may eventually help alleviate some of the issues we presently see with dynamic scaling.

**Limitations**. This work should be considered preliminary; it is not a final format recommendation. To generate results quickly, we used NVIDIA tensor cores with `binary16` accumulators which do not have adequate range for supernormals. In future work, we will use `bfloat16` without tensor core acceleration. Our supernormal implementation presently extends the upper and lower ranges by the same number of binades. Since more range is needed in the lower end than the upper end, we should configure these separately. We also need to modify how DLS works; the present method of using any infinity/NaN to adjust the scale factor aligns the top end of the values to the top end of the format. Instead, scaling should align the mode of the tensor values to a specific 'centering point' in the target format (such as $2^{-5}$).

## References


[1] NVIDIA, "NVIDIA Hopper Architecture In-Depth," https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/, 2022-03-22.

[2] P. Micikevicius, S. Oberman, P. Dubey, M. Cornea *et al.*, "OCP 8-bit floating point specification (OFP8)," Open Compute Project, Tech. Rep., 2023.

[3] IEEE SA P3109 Working Group, "Interim Report on Binary Floating-point Formats for Machine Learning," Oct. 2024, revision 0.9.1.

[4] X. Sun, J. Choi, C.-Y. Chen, N. Wang *et al.*, "Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks," *NeurIPS*, 2019.

[5] B. Noune, P. Jones, D. Justus, D. Masters, and C. Luschi, "8-bit Numerical Formats for Deep Neural Networks," *arXiv:2206.02915*, 2022.

[6] M. Tatsumi, S.-I. Filip, C. White, O. Sentieys, and G. Lemieux, "Mixing Low-Precision Formats in Multiply-Accumulate Units for DNN Training," in *IEEE Int'l Conf. on Field Programmable Tech.*, Dec 2022.

[7] J. Gustafson and I. Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, p. 71–86, Jun. 2017.

[8] P. Micikevicius, S. Narang, J. Alben, G. Diamos *et al.*, "Mixed Precision Training," *arXiv:1710.03740*, 2017.

[9] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea *et al.*, "FP8 formats for deep learning," *arXiv:2209.05433*, 2022.

[10] H. Peng, K. Wu, Y. Wei, G. Zhao *et al.*, "FP8-LM: Training FP8 Large Language Models," *arXiv:2310.18313*, 2023.

[11] S. P. Perez, Y. Zhang, J. Briggs, C. Blake, *et al.*, "Training and inference of large language models using 8-bit floating point," *arXiv:2309.17224*, 2023.

[12] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das *et al.*, "A study of BFLOAT16 for deep learning training," *arXiv:1905.12322*, 2019.

[13] S. Fox, S. Rasoulinezhad, J. Faraone, P. Leong *et al.*, "A Block Minifloat Representation for Training Deep Neural Networks," in *International Conference on Learning Representations*, 2020.

[14] B. D. Rouhani, R. Zhao, A. More, M. Hall *et al.*, "Microscaling Data Formats for Deep Learning," *arXiv:2310.10537*, 2023.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[16] H. Touvron, L. Martin, K. Stone, P. Albert *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv:2307.09288*, 2023.

[17] R. Eldan and Y. Li, "TinyStories: How Small can Language Models be and Still Speak Coherent English?" *arXiv:2305.07759*, 2023.