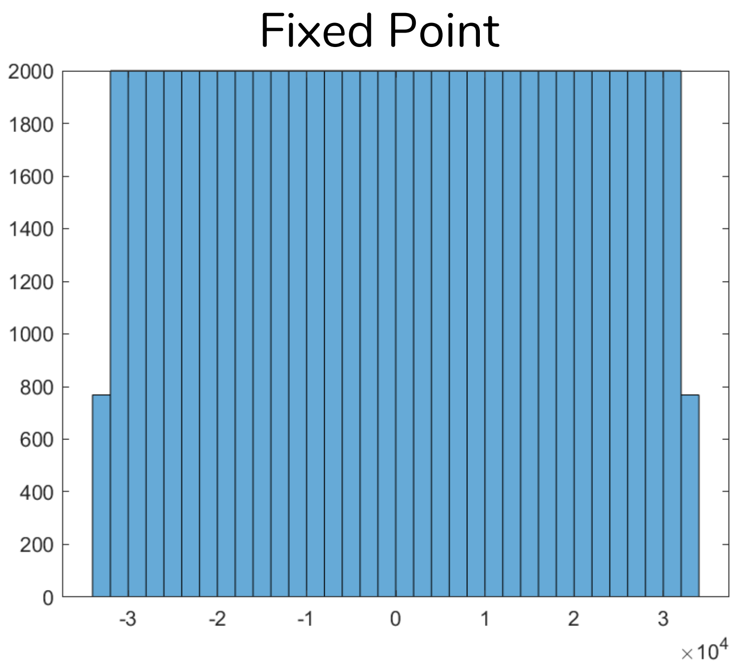
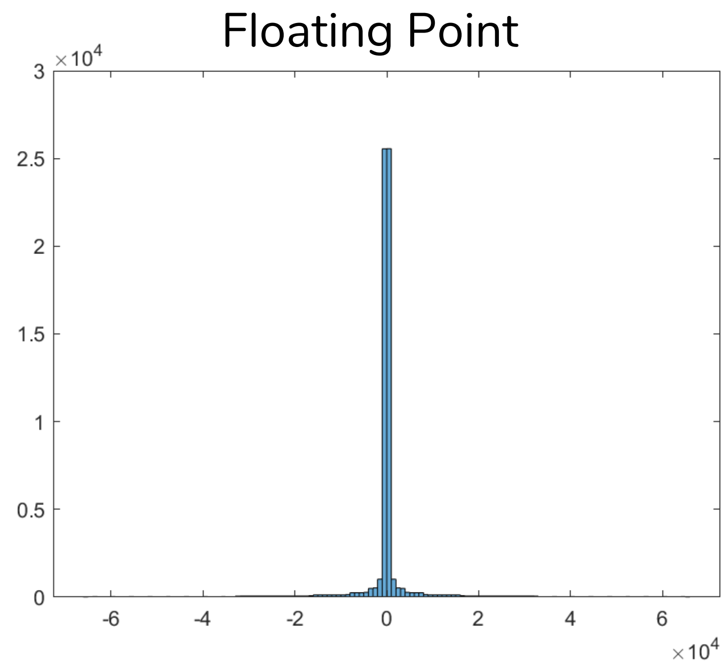
Posits: A Comparison of Number Representations for Hardware Multiply-and-Accumulate Units

Kat Li Yang

Fixed point and floating point are the dominant number representations today

- And for good reason!
- This combination provides balanced precision, range and hardware complexity for general purpose computing





Specialized hardware for deep learning calls for specialized digital arithmetic

- Google
- Facebook
- IBM
- NVIDIA
- Baidu

AI & MACHINE LEARNING

BFloat16: The secret to high performance on Cloud TPUs

POSTED ON NOV 8, 2018 TO AI RESEARCH, DATA INFRASTRUCTURE

Making floating point math highly efficient for AI hardware

AI Hardware

8-Bit Precision for Training Deep Learning Systems

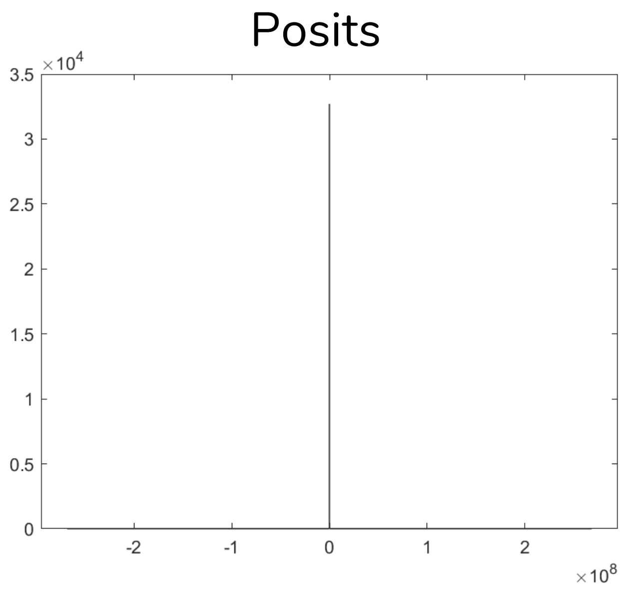
MIXED PRECISION TRAINING

Sharan Narang[*], **Gregory Diamos, Erich Elsen**[†]
Baidu Research
{sharan, gdiamos}@baidu.com

Paulius Micikevicius[*], **Jonah Alben, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, Hao Wu**
NVIDIA
{pauliusm, alben, dagarcia, bginsburg, mhouston, okuchaiev, gavenkatesh, skyw}@nvidia.com

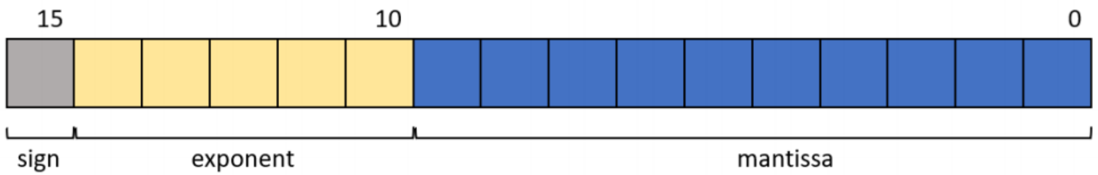
In this study, we examine the feasibility of the (16, 1) Posit as an FP16 alternative

- Posits were introduced by Dr. John Gustafson as a hardware-friendly version of “universal numbers” or “unums”
- Posits have tapered precision, smaller numbers have more fractional bits while larger numbers have fewer



Comparing FP16 to the (16, 1) Posit

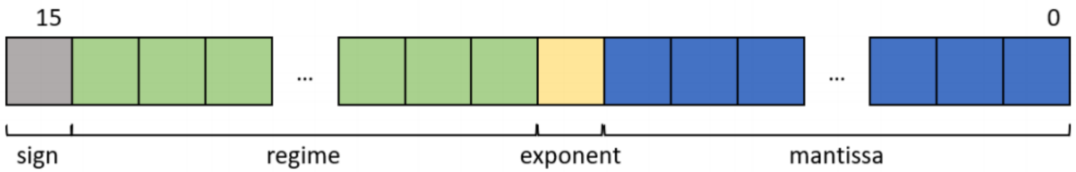
FP16



Bit-width

Exponent width

(16, 1) Posit



Range

-65, 504 to 65, 504

-268, 435, 456 to 268, 435, 456

Wider range

Precision

5.96×10^{-8} to 32

3.73×10^{-9} to 201, 326, 592

More tapered precision

Positive and negative zero and NaN

Single representation of zero and NaN

Decoding the (16, 1) Posit is more complex

`1111100100000000 = -0.01171875`

Take 2's complement `000011100000000`

Sign

Regimen

Exponent

Fraction

$$k = \begin{cases} -m, & \text{with } m \text{ '0's terminated by '1'} \\ m-1, & \text{with } m \text{ '1's terminated by '0'} \end{cases}$$

$$x = (-1)^s \times u^k \times 2^s \times f$$

$$u = 2^{2^{es}}$$

- Bit-wise XOR and adder for 2's complement
- Leading-zero/leading-one detection for regimen decoding
- Variable shift to obtain exponent and fraction

We designed an FP16 MAC and a (16, 1) Posit MAC

- Using Chisel for parametrization and reusability
- Taking inspiration from the HardFloat library to decode the FP16 or Posit into a unified format with zero, NaN, sign, exponent and fraction
- The MAC uses Kulisch accumulation, a wide fixed point accumulator that accommodates the entire range of output values of the product of two floating point or Posit numbers
- Kulisch accumulation enables more precise vector dot products by deferring rounding error to the final output

We constructed and synthesized 4-by-4 systolic arrays using the MAC units

- Internally uses recoded representation, minimizing decoding overhead, especially for Posits
- Uses a weight-stationary dataflow with reference to Gemmini (chosen for simplicity)
- A SystemVerilog testbench was used to test large matrix multiplications by tiling the inputs.

We developed behavioral tests in C++ to test the effect of number representation on neural network accuracy

- Multi-layer perceptron classifies MNIST digits
- 784 input nodes, 64 hidden nodes and 10 output nodes
- Reference 64-bit floating point model achieves ~97% accuracy

Percentage of weight and bias gradients smaller than the smallest representable number in each number representation from the 64-bit model

	FP16	(16, 1) Posit	Reduction
Hidden weights	2.0	1.0	2.0x
Hidden biases	3.6	1.4	2.6x
Output weights	0.0	0.0	-
Output biases	1.9	0.6	3.2x

(16, 1) Posit is less area and power efficient than FP16

Total cell area for 4-by-4 systolic array and constituent modules composed using both FP16 and (16, 1) Posit number representations in μm^2

	FP16	(16, 1) Posit	Increase
Systolic array	75440	105817	40.2%
Systolic array PE	4548	6322	39.0%
MAC unit	3392	4820	42.1%
Multiplier	2662	3653	37.2%
Decoder	183	389	112.5%

Total power consumption of a 4-by-4 systolic array using both FP16 and (16, 1) Posit number representations in W

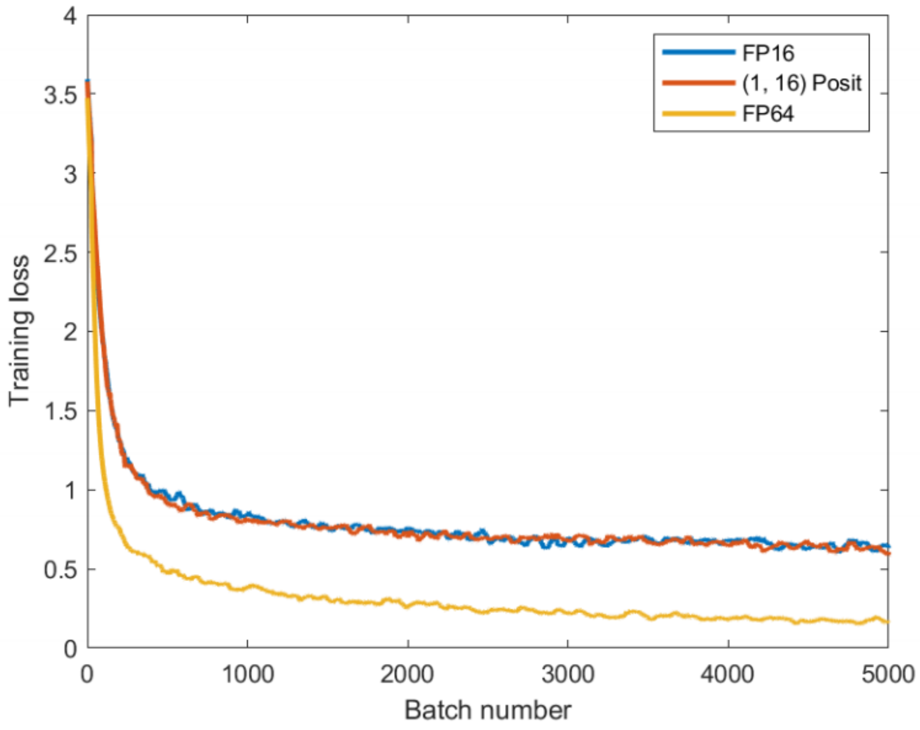
	FP16	(16, 1) Posit	Increase
Leakage	1.90×10^{-3}	2.68×10^{-3}	41.1%
Internal	4.11×10^{-3}	5.53×10^{-3}	34.5%
Switching	1.54×10^{-3}	1.91×10^{-3}	24.0%
Total	7.56×10^{-3}	1.01×10^{-2}	33.6%

For a single-cycle PE, the achievable clock frequency is lower

Highest achievable clock frequency for a 4-by-4 systolic array with a single-cycle data path in each PE for the different number formats

	FP16	(16, 1) Posit
Highest achievable clock frequency	1.25 GHz	1.11 GHz
Time taken for a single matmul	14.4 ns	16.2 ns

Neural network accuracy is comparable but marginally lower



Neural network accuracy trained using different number representations

	FP64	FP16	(16, 1) Posit
Run 1	96.88%	90.01%	90.45%
Run 2	96.89%	90.06%	89.7%
Run 3	96.95%	90.57%	90.32%
Run 4	96.97%	90.17%	89.7%
Run 5	96.97%	91.06%	89.72%
Average	96.93%	90.37%	89.98%

There is no evidence to show that (16, 1) Posit is better than FP16

- Using (16, 1) Posits do not appear to have any advantage over FP16 in deep learning applications
- 40% increase in area, 33% increase in power consumption, longer critical path and lower classification accuracy

Another alternative may be the use of Posits in logarithm arithmetic as suggested by Johnson from Facebook

- Did not attempt to quantify accuracy of this method in this study due to lack of available software libraries but area and power efficiency look promising
- Uses addition to multiply and look-up table to convert to fixed point for accumulation
- Additional reconfigurability in look-up table width and depth

Preliminary synthesis results of a log Posit 4-by-4 systolic array

	(16, 1, 11, 11, 10) log Posit
Area	98290 μm^2
Total Power	7.64×10^{-3}

Application-specific number representation may be the next step

- While floating point remains the most pervasive representation, there are other viable alternatives, but Posits may not be a strong one
- A more nuanced approach to define more application-specific numeric representations/encodings would indubitably benefit future machine learning research