THE POWER OF ARITHMETIC IN ML

by Dr. Alberto A. Del Barrio

University of California at Irvine, 08-jan-2021

Outline

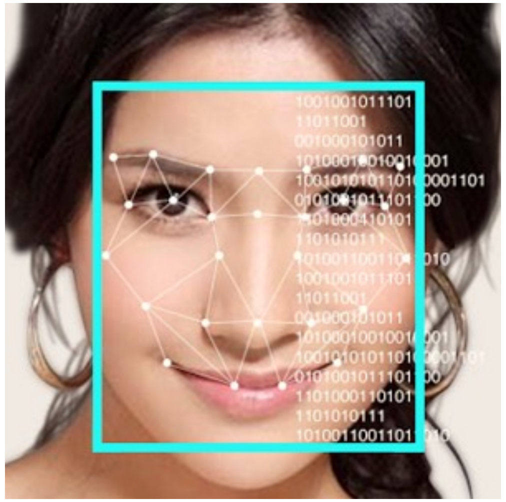
- Deep Learning and Approximate Computing
- Approximate Logarithmic Multiplication
- The Posit Number System
- Conclusions
- Open challenges

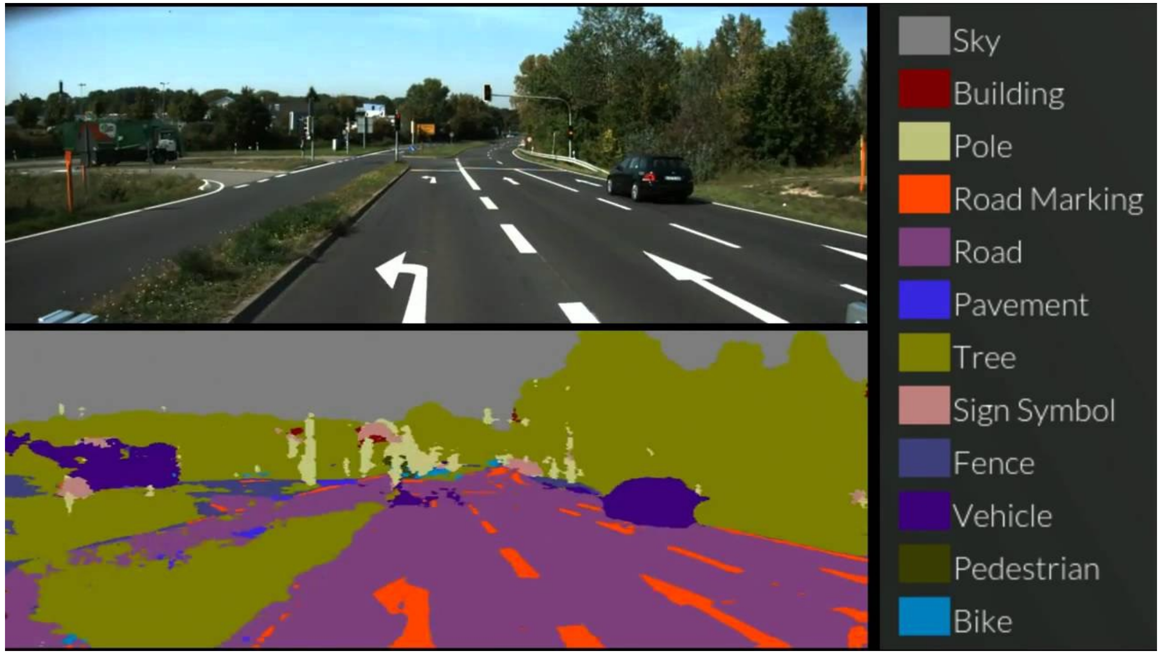
Outline

- <u>**Deep Learning and Approximate Computing**</u>
- Approximate Logarithmic Multiplication
- The Posit Number System
- Conclusions
- Open challenges

Computational Challenge in Machine Learning

- **Machine Learning growing in diverse applications**
  - Autonomous Driving, Face Recognition, Social Analysis…
  - … even for **detecting covid-19**
- **Large amount of data and/or time constraint**
  - Computationally costly and challenging!



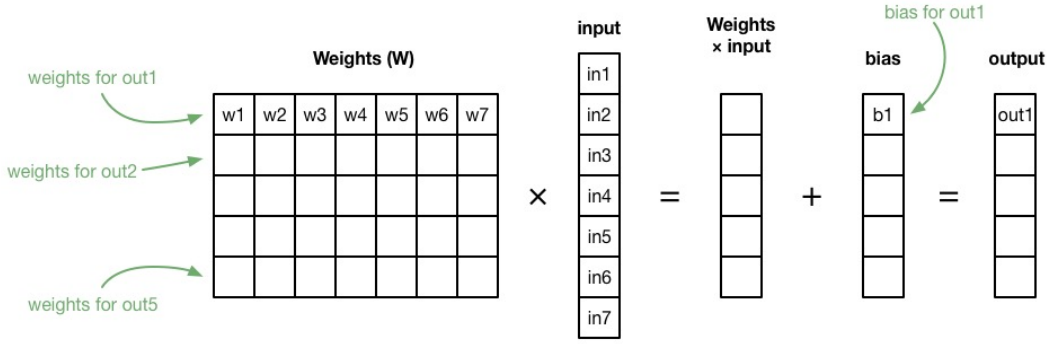


(a) (b) (c)

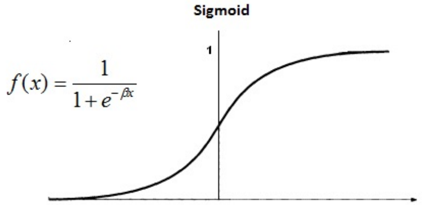
Figure 1. Typical transverse-section CT images: (a) COVID-19; (b) Influenza-A viral pneumonia; (c) no pneumonia manifestations on this chest CT image.

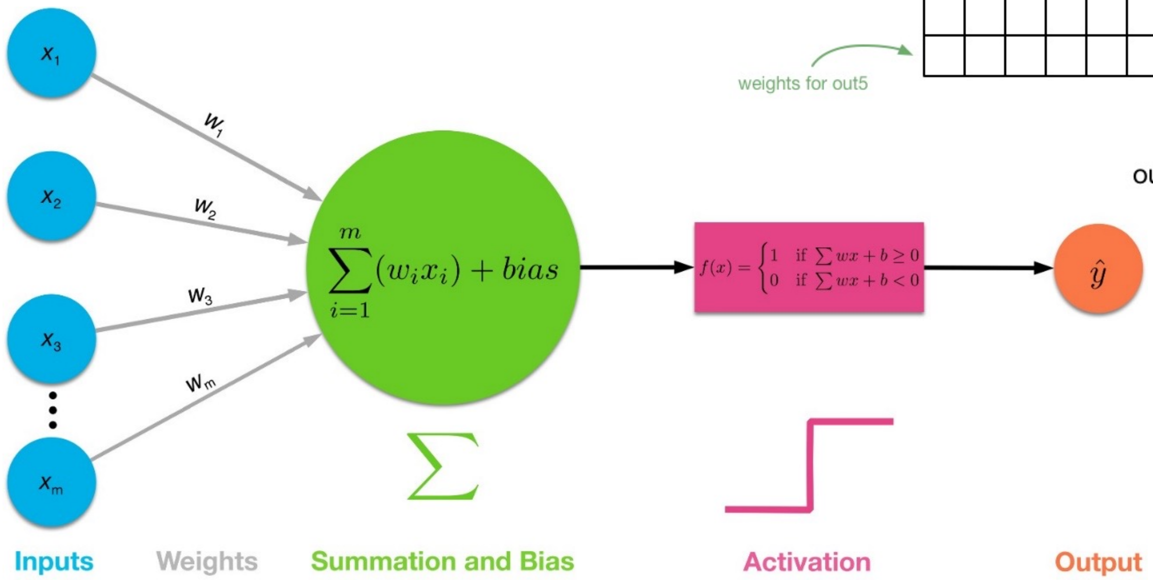
DNN Computation is Mostly Matrix Multiplications

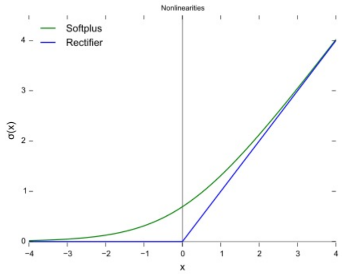
- M by N matrix of weights multiplied by N by 1 vector of inputs
- Need an activation function after this matrix operation: Rectifier, Sigmoid, etc.
- Matrices are dense



output = f(Weights × input + bias)





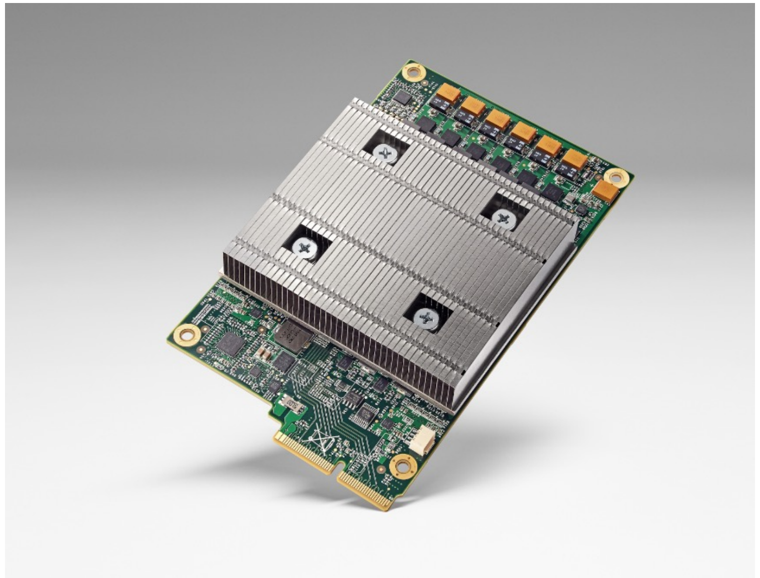


Training and Inference

- **Learning Step:** Weights are produced by training, initially random, using successive approximation that includes backpropagation with gradient descent. Mostly floating point operations. Time consuming
- **Inference Step:** Recognition and classifications. More frequently invoked step. Fixed point operation
- Both steps include many dense matrix vector operations

Opportunities for Power Savings

- **Perfect for hardware acceleration**
  - A lot of MAC operations
  - Parallel and regular structure
- **Suitable for Approximate Computing**
  - Inherent error in machine learning
  - Applications can tolerate small errors
- **Approximate multiplier for the CNN accelerator can reduce power consumption from datacenters to embedded systems**



Google TPU Accelerator

Page Ranking

Translate

Google

Visual Recognition

AlphaGo

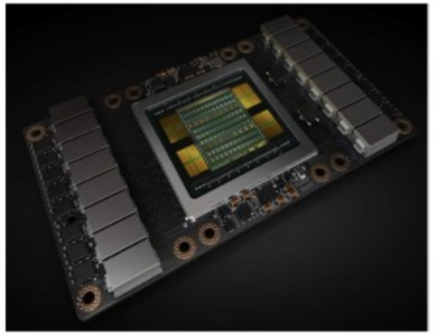
Services that use TPU

Jouppi, Norman P., et al. "In-datacenter performance analysis of a tensor processing unit." Proceedings of the 44th Annual International Symposium on Computer Architecture. ACM, 2017.

Big Players are Investing Heavily on ML

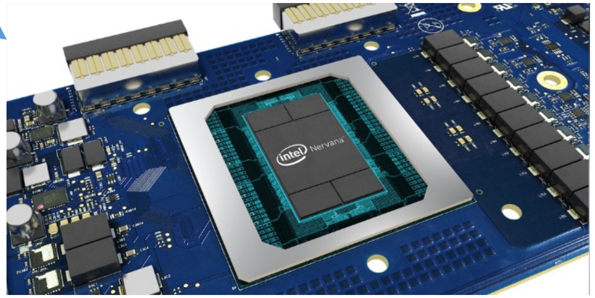
- Custom built chips for AI
  - iPhone X AI Chip
  - Google TPU 2
  - Nvidia acquiring ARM
  - Microsoft Azure and integration of FPGAs
  - Intel acquiring Altera and Nervana; ML accelerator IP
  - AMD acquiring Xilinx
- Software tools
  - Tensorflow, Pytorch, Caffe
  - ML algorithms

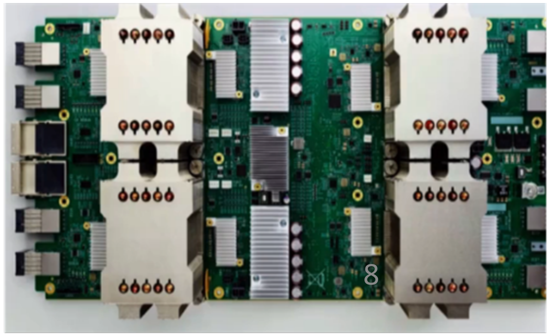
Nvidia Volta GV100 (2017)



- 15 FP32 TFLOPS
- 120 Tensor TFLOPS
- 16GB HBM2 @ 900GB/s
- 300W TDP
- 12nm process
- 21B Transistors
- die size: 815 mm2
- 300GB/s NVLink

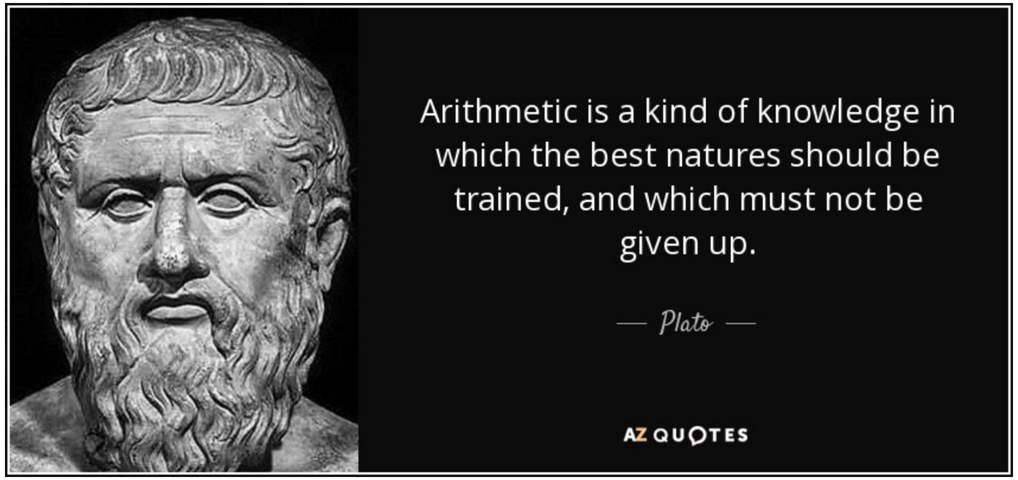
❑ New Generation of Accelerators are able to train and inference in one chip



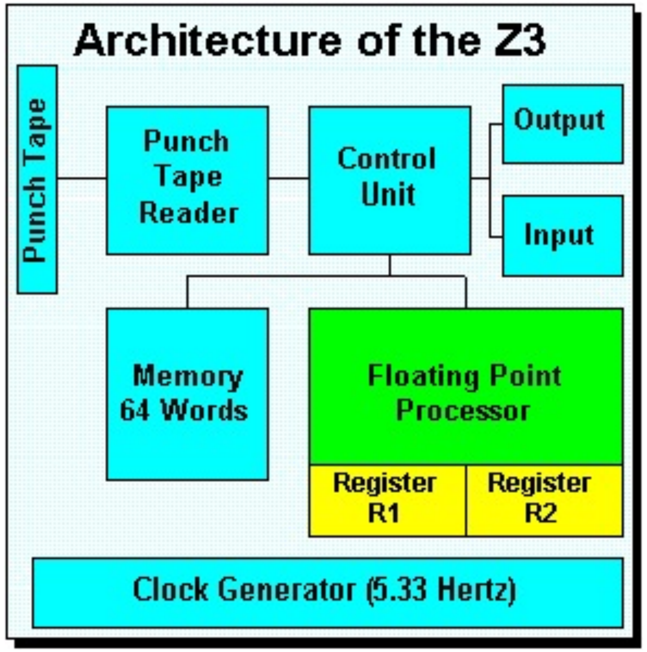


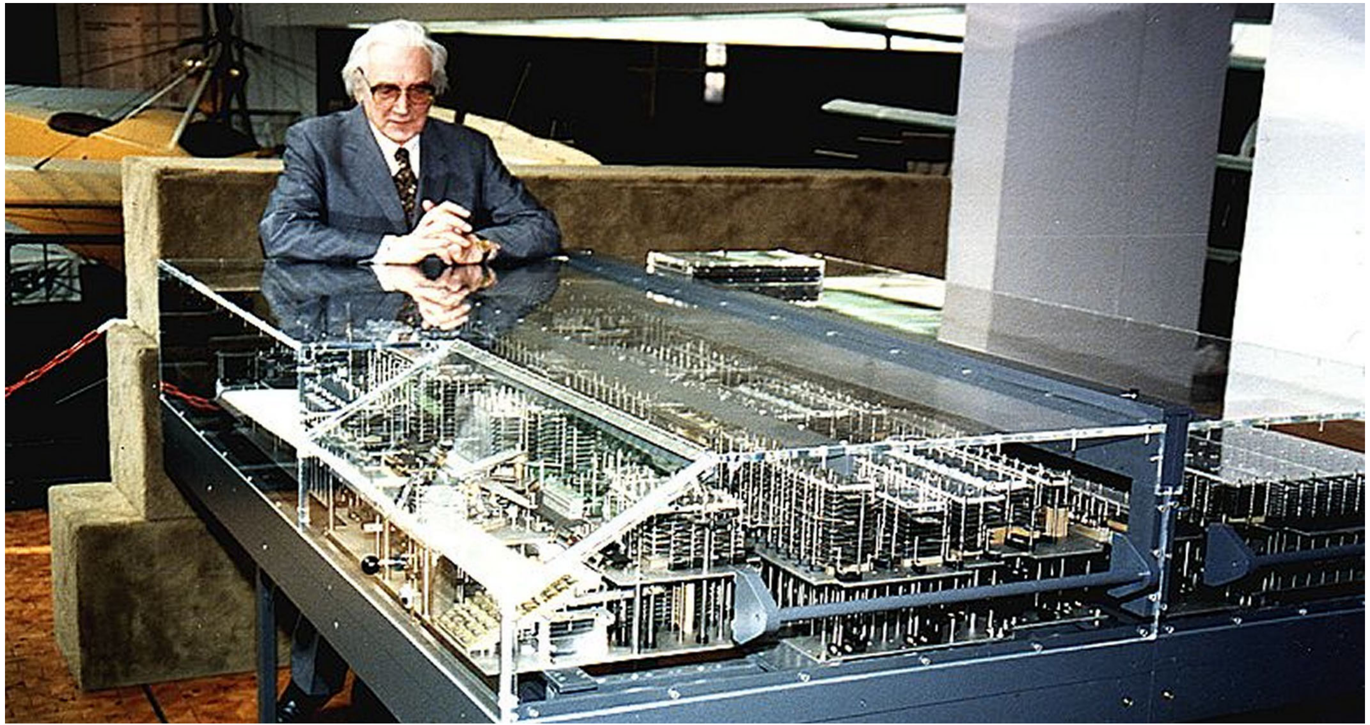
Addition is deeply optimized …

- **… I just type +**
- Why on earth should I learn about Arithmetic? I prefer Python
- This is something really ancient



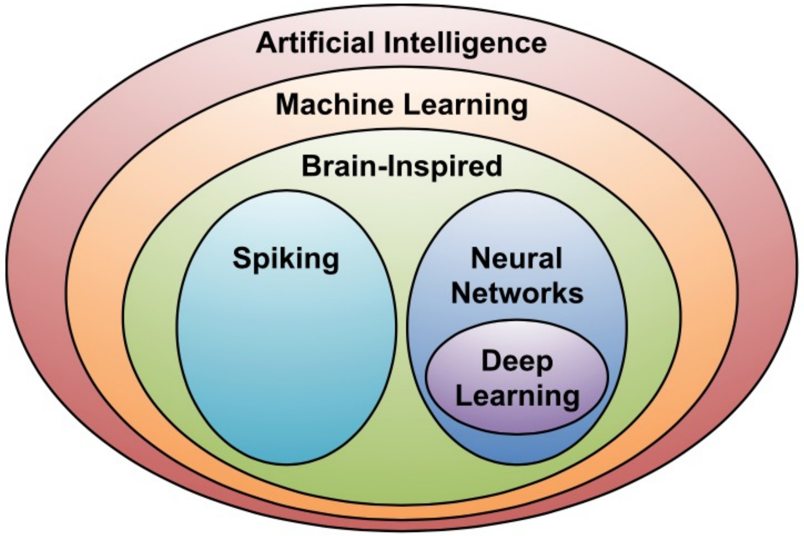
1941

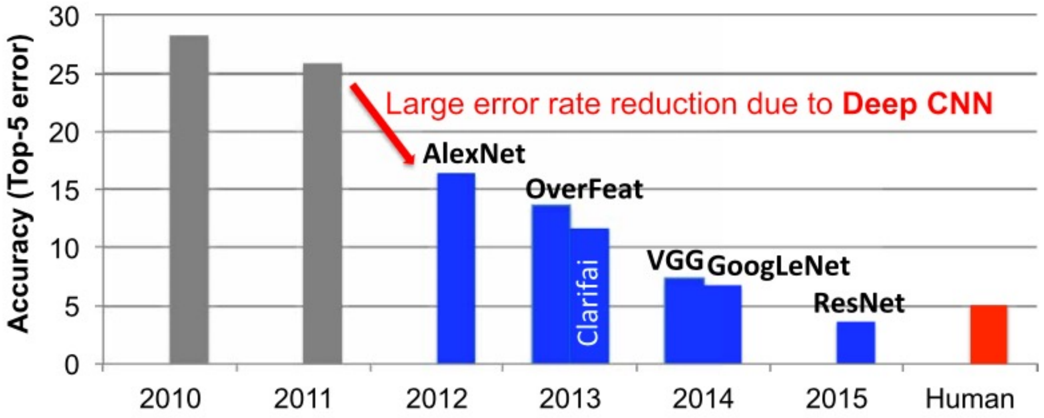




Addition is deeply optimized …

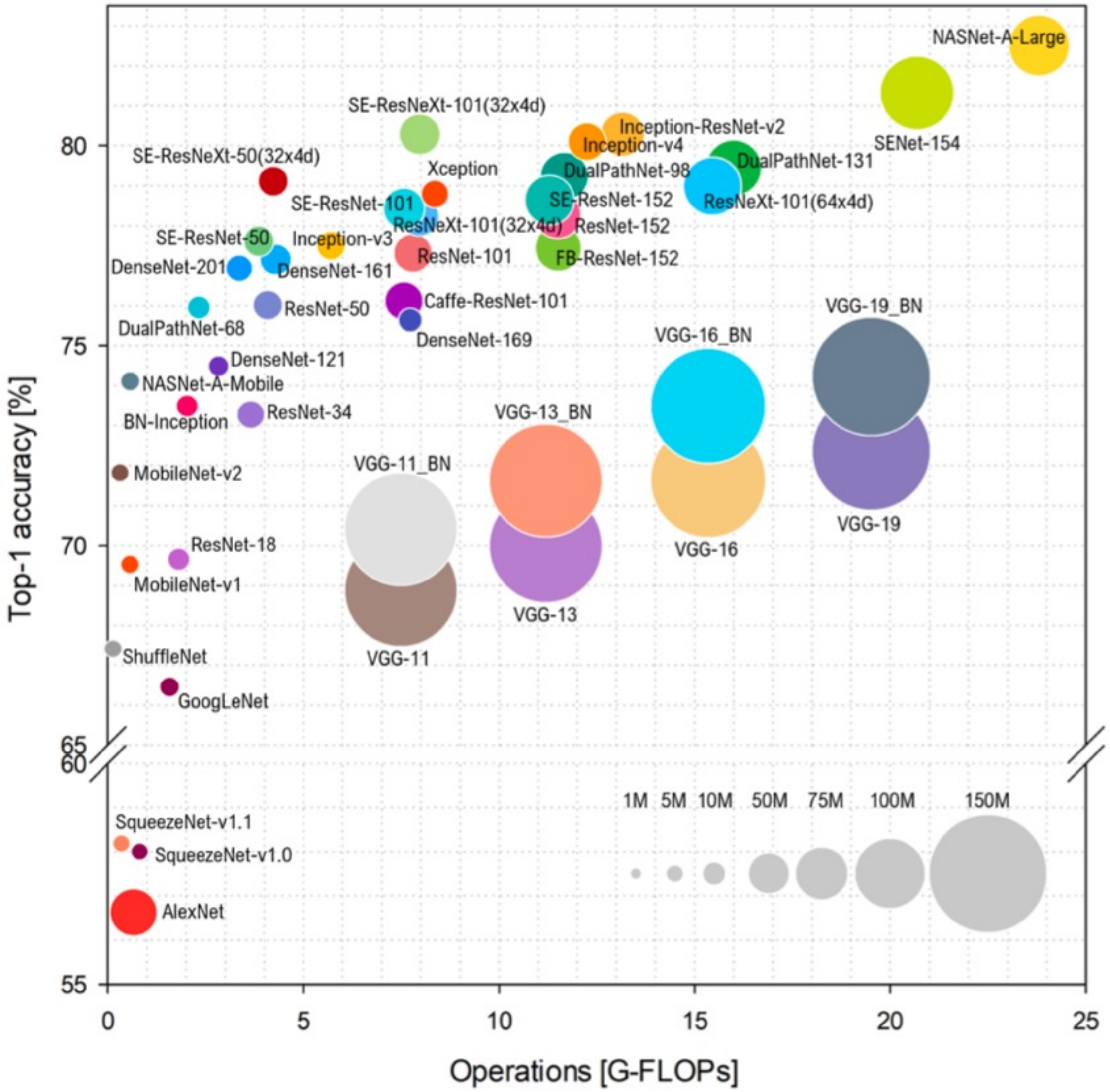
- **… I just type +**
- Neural Networks theory was developed in the mid 20th century
- Until we did not have enough computational power and available data (around 2010), they did not take off





V. Sze, Y. Chen, T. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295-2329, Dec. 2017, doi: 10.1109/JPROC.2017.2761740.

Some figures



- DNNs are very complex
- The number of parameters is usually larger than 10M
- Training is very expensive
- *Jevons Paradox* or "dying because of the success"

Consumption	CO_2e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

Table 1: Estimated CO_2 emissions from training common NLP models, compared to familiar consumption.[1]

[1] S. Bianco, R. Cadene, L. Celona and P. Napoletano, "Benchmark Analysis of Representative Deep Neural Network Architectures," in *IEEE Access*, vol. 6, pp. 64270-64277, 2018, doi: 10.1109/ACCESS.2018.2877890.

Some figures

Benchmark	Error rate	Polynomial			Exponential		
		Computation Required (Gflops)	Environmental Cost (CO_2)	Economic Cost ($)	Computation Required (Gflops)	Environmental Cost (CO_2)	Economic Cost ($)
ImageNet	Today: 11.5%	10^{14}	10^{6}	10^{6}	10^{14}	10^{6}	10^{6}
	Target 1: 5%	10^{19}	10^{10}	10^{11}	10^{27}	10^{19}	10^{19}
	Target 2: 1%	10^{28}	10^{20}	10^{20}	10^{120}	10^{112}	10^{112}
MS COCO	Today: 46.7%	10^{14}	10^{6}	10^{6}	10^{15}	10^{7}	10^{7}
	Target 1: 30%	10^{23}	10^{14}	10^{15}	10^{29}	10^{21}	10^{21}
	Target 2: 10%	10^{44}	10^{36}	10^{36}	10^{107}	10^{99}	10^{99}
SQuAD 1.1	Today: 4.621%	10^{13}	10^{4}	10^{5}	10^{13}	10^{5}	10^{5}
	Target 1: 2%	10^{15}	10^{7}	10^{7}	10^{23}	10^{15}	10^{15}
	Target 2: 1%	10^{18}	10^{10}	10^{10}	10^{40}	10^{32}	10^{32}
CoLLN 2003	Today: 6.5%	10^{13}	10^{5}	10^{5}	10^{13}	10^{5}	10^{5}
	Target 1: 2%	10^{43}	10^{35}	10^{35}	10^{82}	10^{73}	10^{74}
	Target 2: 1%	10^{61}	10^{53}	10^{53}	10^{181}	10^{173}	10^{173}
WMT 2014 (EN-FR)	Today: 54.4%	10^{12}	10^{4}	10^{4}	10^{12}	10^{4}	10^{4}
	Target 1: 30%	10^{23}	10^{15}	10^{15}	10^{30}	10^{22}	10^{22}
	Target 2: 10%	10^{43}	10^{35}	10^{35}	10^{107}	10^{99}	10^{100}

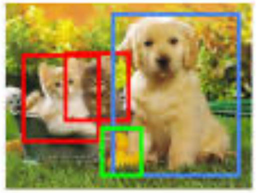
Figure 4: Implications of achieving performance benchmarks on the computation (in Gigaflops), carbon emissions (lbs), and economic costs ($USD) from deep learning based on projections from polynomial and exponential models. The carbon emissions and economic costs of computing power usage are calculated using the conversions from [82]

Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, Gabriel F. Manso: The Computational Limits of Deep Learning. CoRR abs/2007.05558 (2020)

Classification

CAT

Object Detection



CAT, DOG, DUCK

ImageNet

Some figures

Nowadays these are toy examples

Metrics	LeNet 5	AlexNet	Overfeat fast	VGG 16	GoogLeNet v1	ResNet 50
Top-5 error†	n/a	16.4	14.2	7.4	6.7	5.3
Top-5 error (single crop)†	n/a	19.8	17.0	8.8	10.7	7.0
Input Size	28×28	227×227	231×231	224×224	224×224	224×224
# of CONV Layers	2	5	5	13	57	53
Depth in # of CONV Layers	2	5	5	13	21	49
Filter Sizes	5	3,5,11	3,5,11	3	1,3,5,7	1,3,7
# of Channels	1, 20	3-256	3-1024	3-512	3-832	3-2048
# of Filters	20, 50	96-384	96-1024	64-512	16-384	64-2048
Stride	1	1,4	1,4	1	1,2	1,2
Weights	2.6k	2.3M	16M	14.7M	6.0M	23.5M
MACs	283k	666M	2.67G	15.3G	1.43G	3.86G
# of FC Layers	2	3	3	3	1	1
Filter Sizes	1,4	1,6	1,6,12	1,7	1	1
# of Channels	50, 500	256-4096	1024-4096	512-4096	1024	2048
# of Filters	10, 500	1000-4096	1000-4096	1000-4096	1000	1000
Weights	58k	58.6M	130M	124M	1M	2M
MACs	58k	58.6M	130M	124M	1M	2M
Total Weights	60k	61M	146M	138M	7M	25.5M
Total MACs	341k	724M	2.8G	15.5G	1.43G	3.9G
Pretrained Model Website	[56]‡	[57, 58]	n/a	[57–59]	[57–59]	[57–59]

- Inference is not easy either

V. Sze, Y. Chen, T. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295-2329, Dec. 2017, doi: 10.1109/JPROC.2017.2761740.

Some figures

- ImageNet validation dataset (50,000 images)
- What if I save 1 pJ per multiplication? (just in inference)
  - LeNet-5 ➔ 341 kmults/inference * 1 pJ * 50 kinferences = 17.1 mJ
  - AlexNet ➔ 724 Mmults/inference * 1 pJ * 50 kinferences = 36.2 J
  - VGG-16 ➔ 15.5 Gmults/inference * 1 pJ * 50 kinferences = 775 J
  - GoogleLeNet v1 ➔ 1.43 Gmults/inference * 1 pJ * 50 kinferences = 71.5 J
  - ResNet-50 ➔ 3.9 Gmults/inference * 1 pJ * 50 kinferences = 195 J

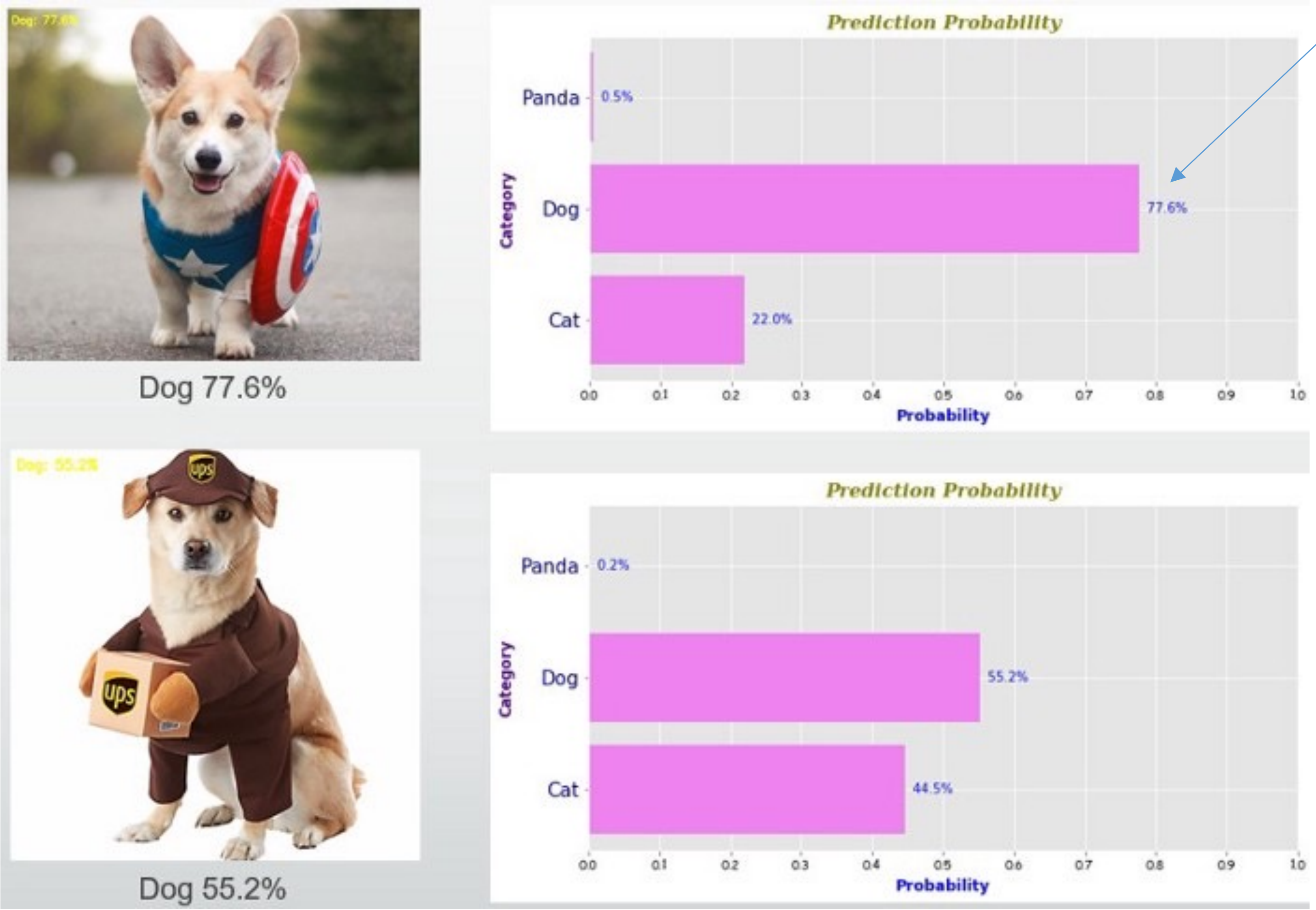
Some figures

- iPhone 12 Pro, USB-C 20W adapter, 2h/full charge ➔ 20W * 2h* 3600s/h = 144 kJ
- Average USA house consumption per year [1] ➔ 10,649 kWh = 10,649 * 1,000 W/1kW * 3600s/1h = 38336.4 MJ
- How many people work on ML (particularly DNNs)?
  - Saving 1 pJ per multiplication you could be charging your iPhone 12 … like forever ☺
- **Jevons Paradox.** Let’s say, people validating Imagenet on ResNet-50
  - 100 people ➔ 100 * 195 J = 0.0195 MJ
  - 1,000 people ➔ 1,000 * 195 J = 0.195 MJ
  - 10,000 people ➔ 1,0000 * 195 J = 1.95 MJ
  - 100,000 people ➔ 100,000 * 195 J = 19.5 MJ
- Usually researchers make mistakes, so they will repeat the tests and also will test other NNs (just for fun ☺)

[1] https://www.eia.gov/tools/faqs/faq.php?id=97&t=3

So why Approximate Computing?

- DNN outputs are probabilities. It does not matter the value, the only thing that matters is the relative order



If instead of 77.6%, we had 70% or even 50%, it would not matter

But be careful with the approximations !!!

Outline

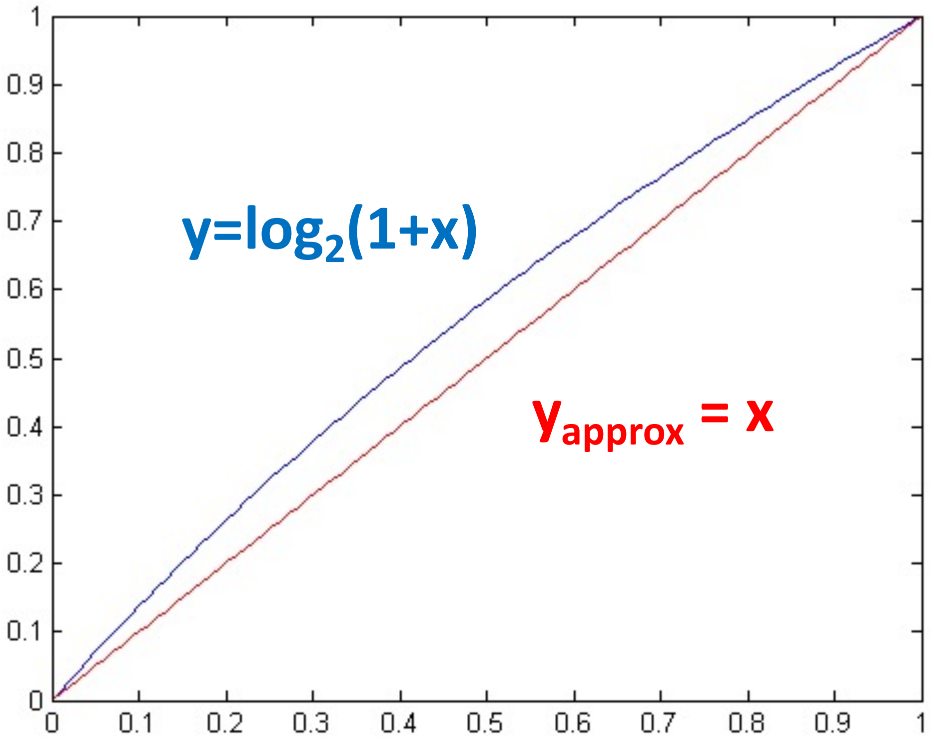
- Deep Learning and Approximate Computing
- **Approximate Logarithmic Multiplication**
- The Posit Number System
- Conclusions
- Open challenges

Approximate Log Multiplication

- Arithmetic is something very ancient … so let's try something very ancient
- Logarithms were defined by Burgi and Napier at the beginning of the XVII century
- Current logarithms and their connection with the exponential were defined by Euler in the XVIII century
- **Multiplication → Addition in log domain, log(A*B) = log(A) + log(B)**

Approximate Log Multiplication

- Mitchell introduced digital logarithmic multiplication and division in 1962
- Based on approximating *$log_2(1+x)$* with *x*, when *x* belongs to [0,1)



$$Z = \sum_{i=0}^{n-1} 2^i z_i = 2^k \left(1 + \sum_{i=j}^{k-1} 2^{i-k} z_i\right), \quad k \geq j \geq 0.$$

$$\log_2(Z) = \log_2\left(2^k\left(1 + \sum_{i=j}^{k-1} 2^{i-k} z_i\right)\right) = k + log_2(1+x)$$

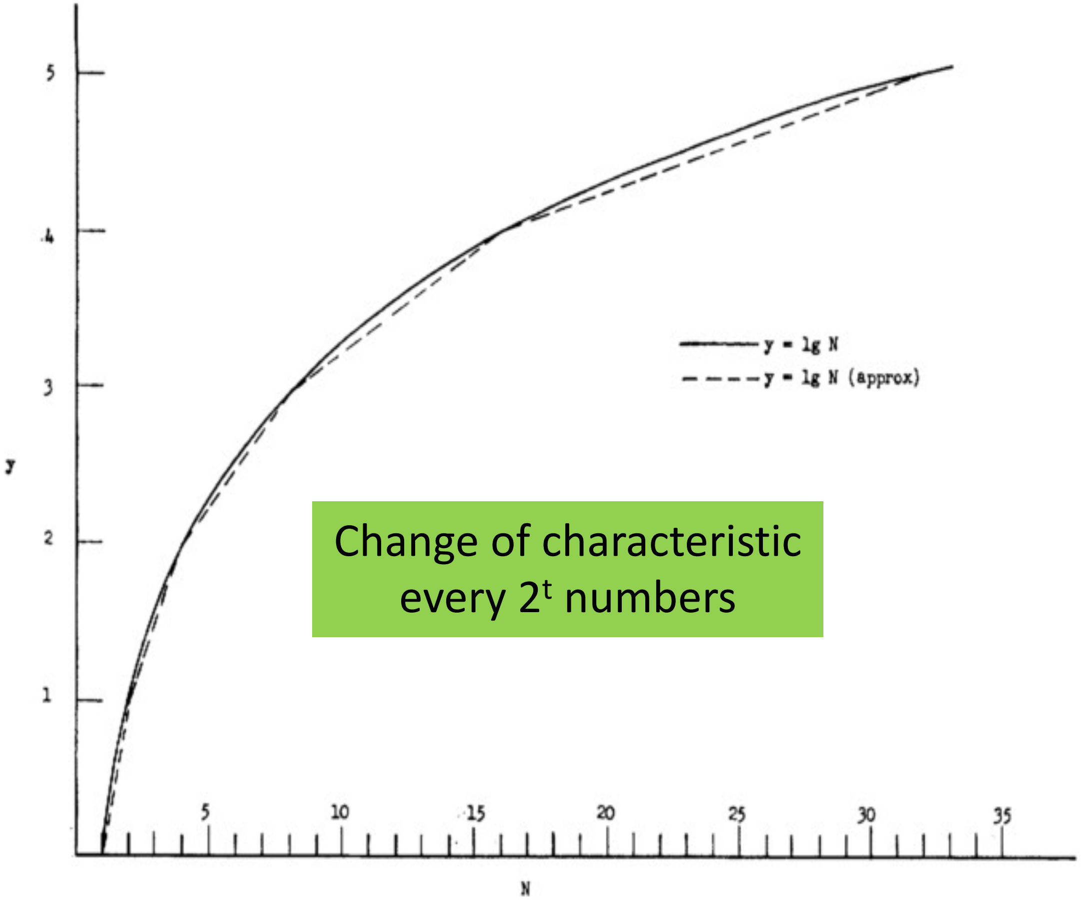
characteristic **fraction**

Mitchell, J. N. (1962). Computer Multiplication and Division Using Binary Logarithms. *Electronic Computers, IRE Transactions on*, *EC-11*(4), 512–517. http://doi.org/10.1109/TEC.1962.5219391

Approximate Log Multiplication

- **Based on the approximate logarithm**
- Reduces logarithm to Leading One Detector (**LOD**) and Shifter operations

A (binary)	Approx. log(A) (binary)
00001	000.0000
00010	001.0000
00011	001.1000
00100	010.0000
00101	010.0100
00110	010.1000
00111	010.1100
01000	011.0000
…	…
10000	100.0000



Mitchell, J. N. (1962). Computer Multiplication and Division Using Binary Logarithms. *Electronic Computers, IRE Transactions on*, *EC-11*(4), 512–517. http://doi.org/10.1109/TEC.1962.5219391

Approximate Log Multiplication

- **Worst case relative error = 11.1%**

In the log domain, a number A = $2^{k_A} * (1 + x_A)$

	7	6	5	4	3	2	1	0	
A	0	0	0	0	1	1	1	1	15
B	0	0	0	0	0	0	1	1	3

k_A (characteristic of A)

x_A (mantissa of A)

C=log2(A)= (3 << 7) & ((A – 2^3) << 4)	0	1	1	1	1	1	0	0	0	0
D=log2(B)= (1 << 7) & ((B - 2^1) << 6)	0	0	1	1	0	0	0	0	0	0

Approx: 3.875 Exact: 3.907

Approx: 1.500 Exact: 1.585

Shifts, masks and concatenations

C+D = E = 5.375	0	1	0	1	0	1	1	0	0	0	0

F= Antilog2(E) = $2^{k_E} * (1 + x_E)$ = 44

										7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0

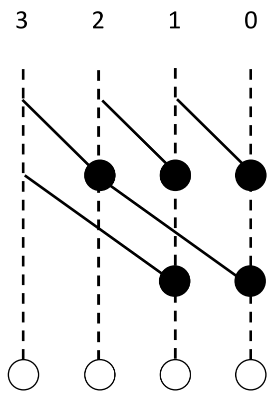
44

2% error

F= $2^5 * (1 + 0.375)$ = 1.01100 << *5* = 44

Mitchell Log Multiplier

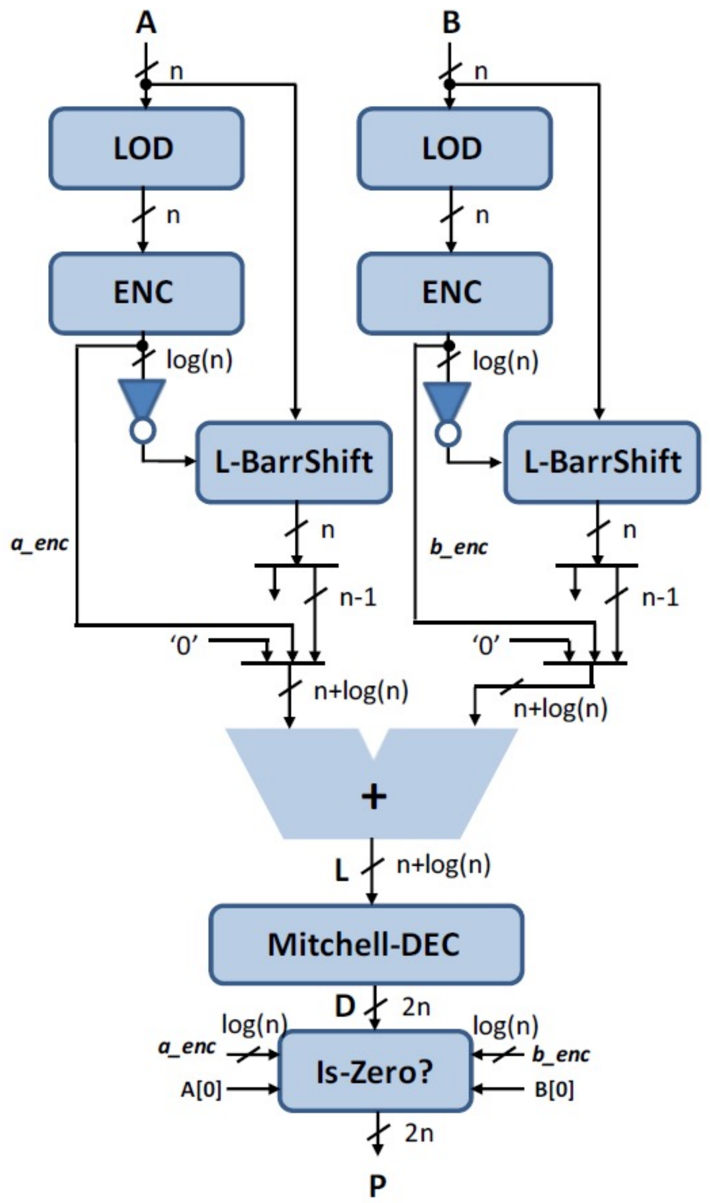
- **Logic optimization of LOD and ENC**
  - **Fast and efficient fully parallel LOD**
    - **One-hot output**
  - OR-Tree encoder. e.g. 113 = 0111 0001
    - One-hot LOD ($h_7h_6h_5h_4h_3h_2h_1h_0$) ➔ 0100 0000
    - Or-Tree encoder ($e_2e_1e_0$) ➔ 110
    - $e_2 = h_7$ or h_6 or h_5 or h_4, $e_1 = h_7$ or h_6 or h_3 or h_2
- **Shift amount calculation**
  - (n-k-1) = not(k) when n is a power of 2



● $= m_{i-1,j} + m_{i-1,j+2^{i-1}}$

○ $= h_j = \begin{cases} z_j & j = n-1 \\ \overline{m_{\log(n),j+1}} \cdot z_j & j < n-1 \end{cases}$

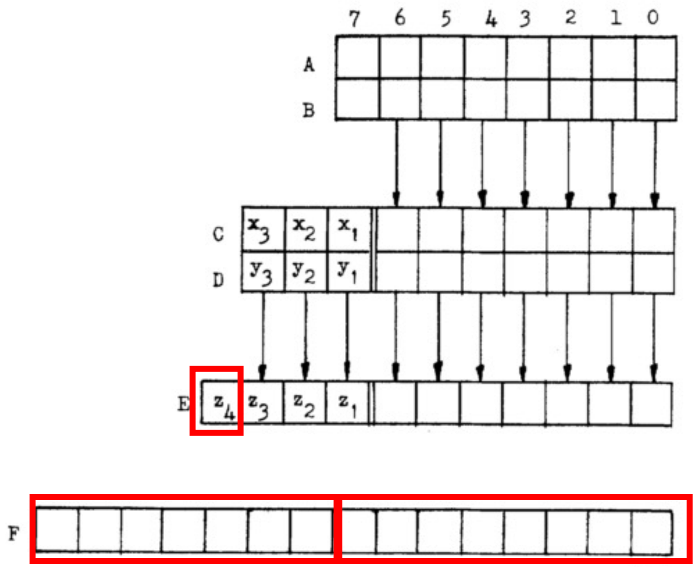
4-bit parallel LOD



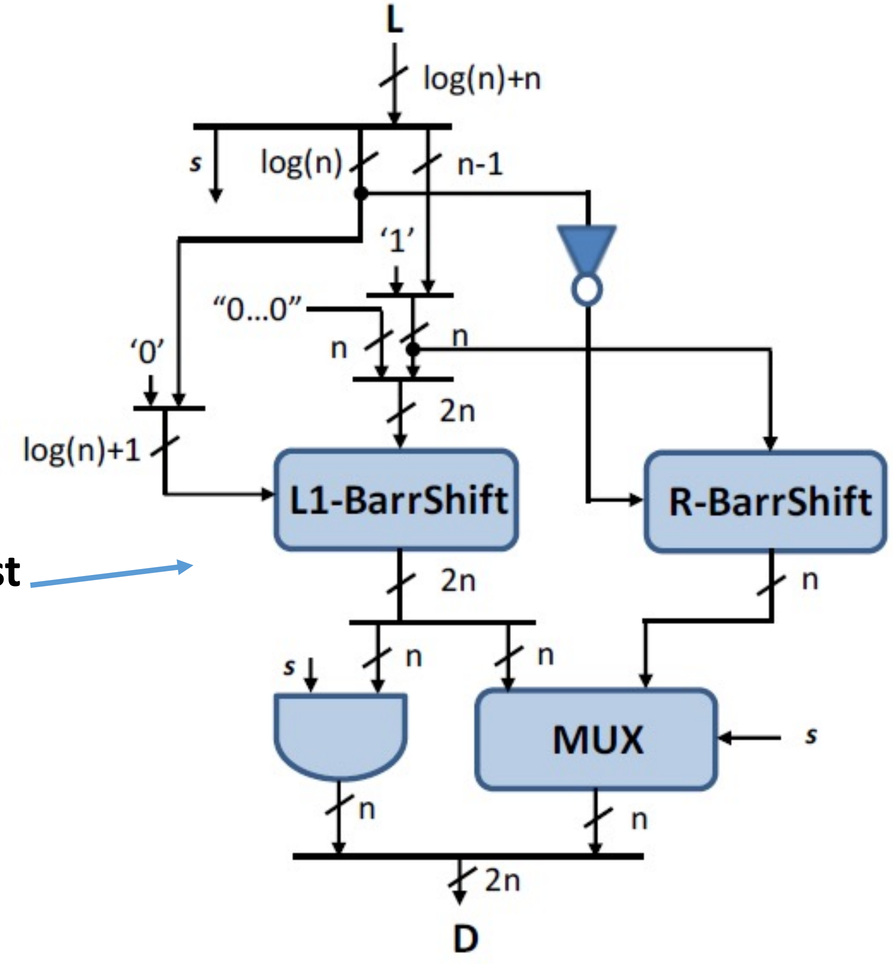
M. S. Kim, A. A. Del Barrio, R. Hermida and N. Bagherzadeh, "Low-power implementation of Mitchell's approximate logarithmic multiplication for convolutional neural networks," ASP-DAC, Jeju, 2018, pp. 617-622. doi: 10.1109/ASPDAC.2018.8297391

Mitchell Decoder

- Given the characteristic (k), a normalized mantissa ($x \in [0,1)$), $X = 2^k*(1+x)$
- Two cases for decoding
  - Large Characteristic (msb = 1)
    - L1 barrel shifter ➔ shamtL = (k and ($2^{log(n)}$-1)) + 1
    - L1 is a customized left shifter that performs the +1 for free
  - Small Characteristic (msb = 0)
    - Right barrel shifter ➔ shamtR = not(k and ($2^{log(n)}$-1))
- Only AND needed for MSBs



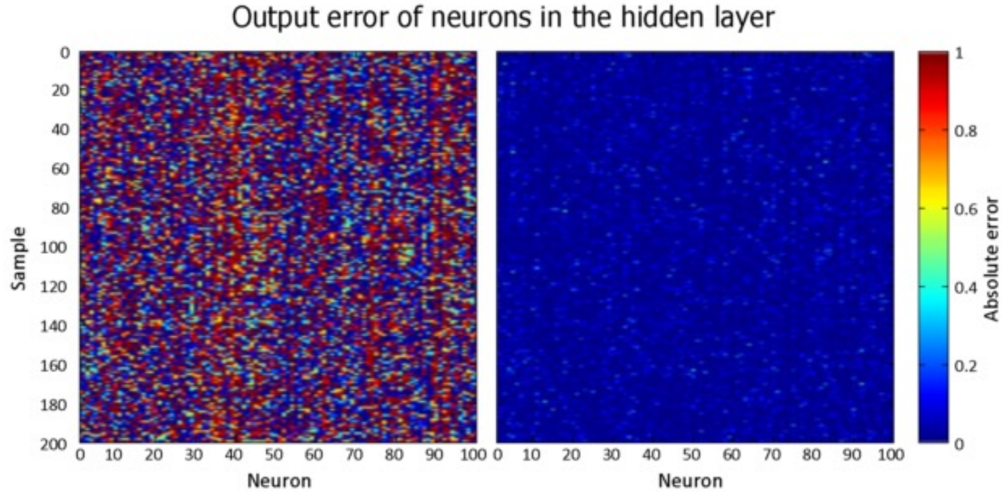
This is the most complex block within the multiplier

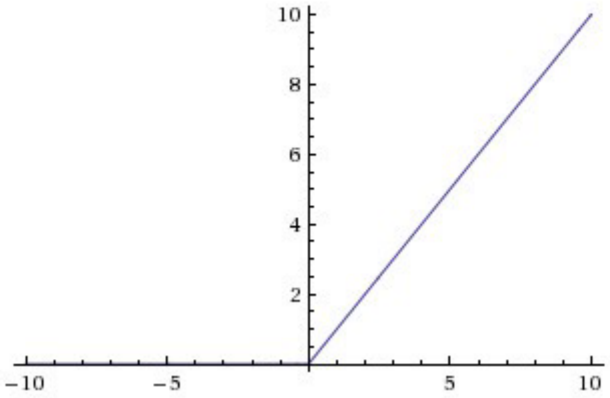


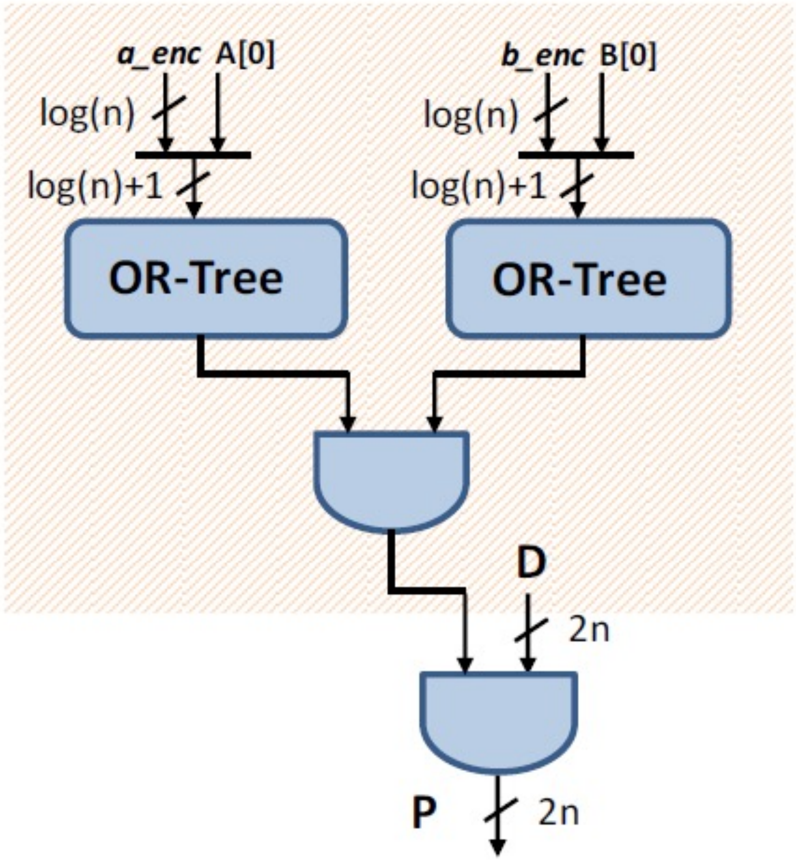
Zero Detection Unit

- **Critical to CNN accuracy**

Quick check: if the characteristic is zero and the lsb is zero, the operand is zero





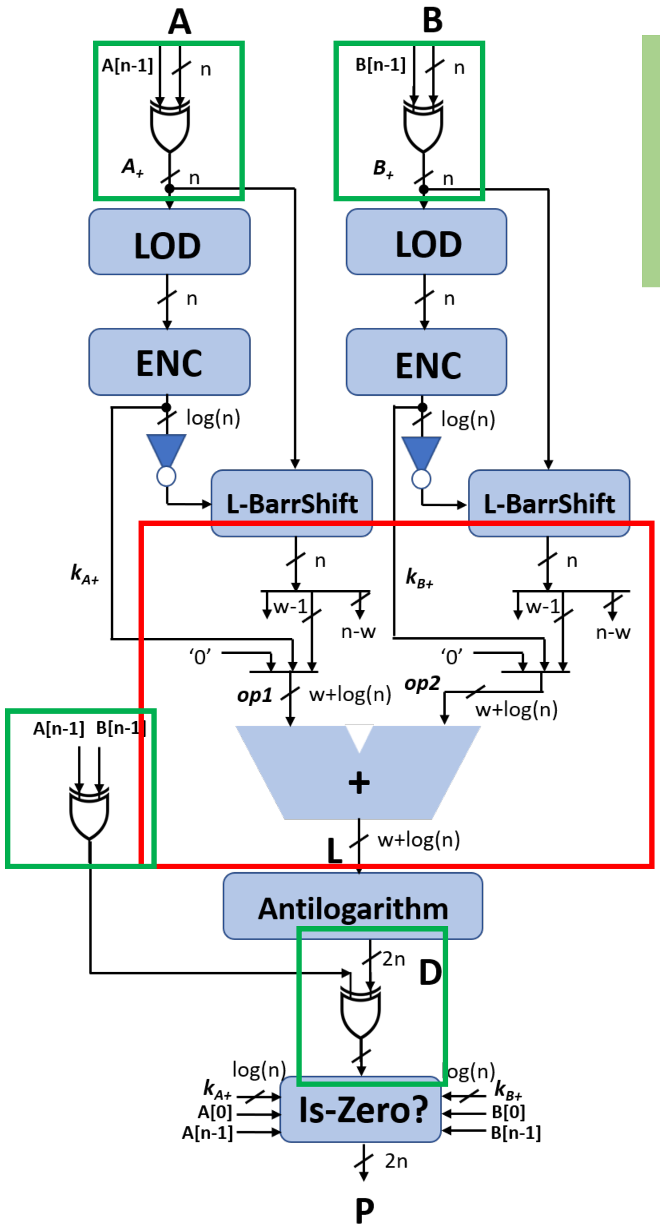


Mrazek, V., Sarwar, S. S., Sekanina, L., Vasicek, Z., & Roy, K. (2016). Design of power-efficient approximate multipliers for approximate artificial neural networks. *Proceedings of the 35th International Conference on Computer-Aided Design - ICCAD '16*, 1–7.

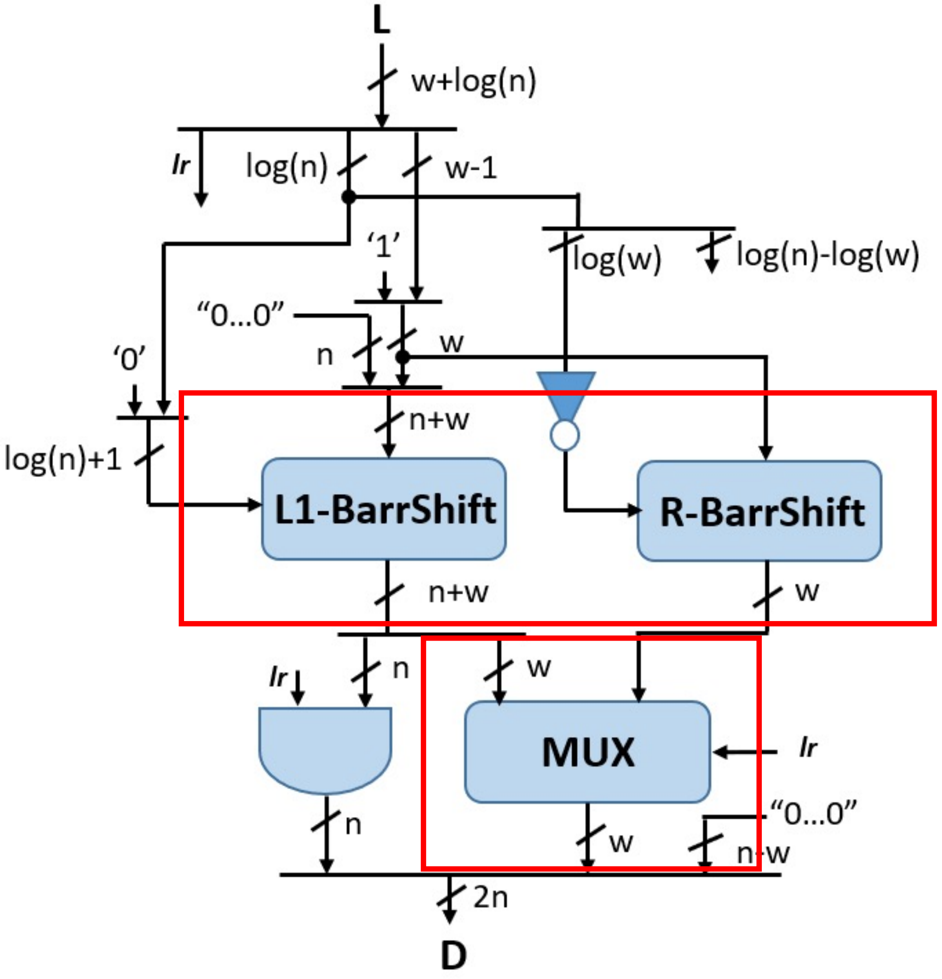
Some refinements

- Negative handling
- In the Logarithmic Number System (LNS), the weight/relevance of the characteristic with respect to the mantissa is very large
    - +1 in the characteristic is equivalent to double the value of the number
- Maybe not all the bits of the mantissa are necessary for a good enough implementation
- Maybe it is possible to further reduce the error

Some refinements

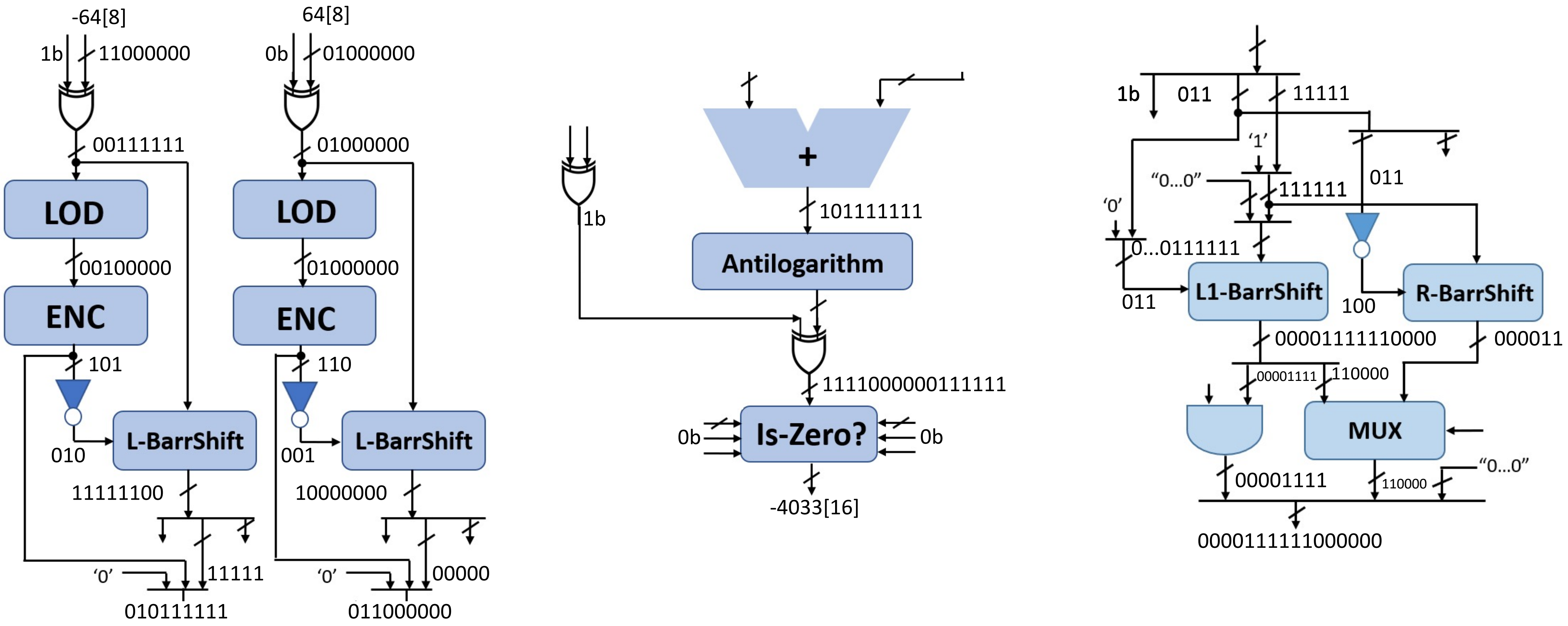


C1 conversion for handling negative numbers



The size of the shifters is reduced from *2n* to *n+w* and from n to *w*

Approximate Log Multiplier Animation



Energy and accuracy

- 32nm, 250 MHz clock

	N=16					N=32				
	Fixed-point	Mitch-$w8$	2-Stage Iter. Log.	1-Alphabet ASM	DRUM6	Fixed-point	Mitch-$w8$	2-Stage Iter. Log.	1-Alphabet ASM	DRUM6
Crit. Path (ns)	2.23	1.90	3.88	2.64	2.64	3.78	2.50	4.00	4.00	3.96
Area (um^2)	2032	1135	3335	1543	1375	8627	2092	11218	7642	2917
Tot. Power (mW)	1.24	0.61	1.79	1.04	0.88	6.02	1.08	5.70	5.81	1.54
Energy (pJ)	2.77	1.16	6.95	2.75	2.32	22.76	2.70	22.80	23.24	6.10
Energy Savings	0%	58%	-151%	1%	16%	0%	88%	0%	-2%	73%

No accuracy degradation in ImageNet + AlexNet

	Fixed	Mitch-$w8$	IterLog2	ASM	DRUM6
Top-1	58.3%	58.2%	58.2%	41.6%	58.2%
Top-5	80.2%	80.2%	80.2%	67.0%	80.2%

M. S. Kim, A. A. Del Barrio, L. T. Oliveira, R. Hermida and N. Bagherzadeh, "Efficient Mitchell's Approximate Log Multipliers for Convolutional Neural Networks," in IEEE Transactions on Computers. doi: 10.1109/TC.2018.2880742
M. S. Kim, A. A. Del Barrio, R. Hermida, N. Bagherzadeh:
Low-power implementation of Mitchell's approximate logarithmic multiplication for convolutional neural networks. ASP-DAC 2018: 617-622

Approximate Log Multiplier: wrapping up

- Saves up to 91% power at 32 bits vs. exact fixed-point multiplier
- Minimal classification accuracy degradations on CNNs

Synthesis Results

	Fixed	Mitch-w6
Cell Area (um^2)	8627	1815
Critical Path (ns)	3.78	2.19
Power (mW)	6.02	0.9
Energy (pJ)	22.76	1.97
Area Savings		**79.0 %**
Energy Savings		**91.3 %**

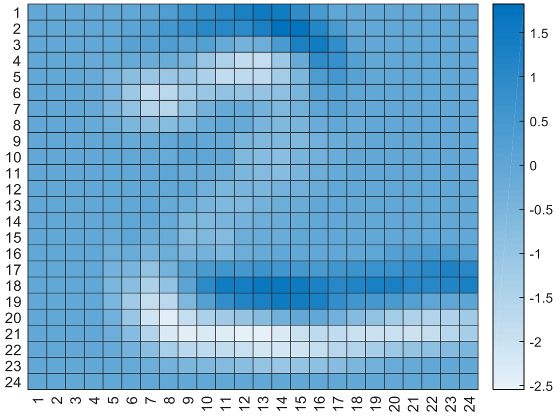
CNN Image Classification Accuracy

Dataset	Fixed	Mitch-w6
MNIST (LeNet)	99.0 %	99.0 %
CIFAR-10 (Cuda-convnet)	81.4 %	81.3 %
Top-1 ImageNet (AlexNet)	56.8 %	56.5 %
Top-5 ImageNet (AlexNet)	79.9 %	79.8 %

Approximate Log Multiplier: wrapping up

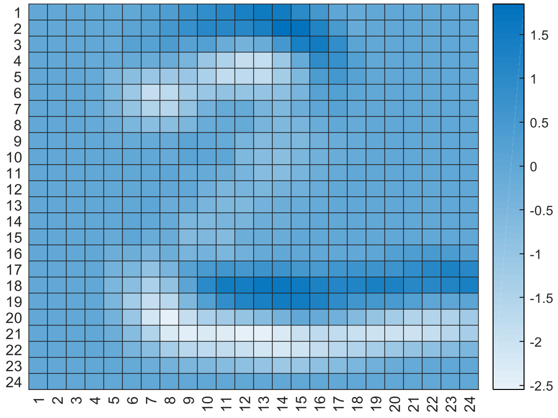
- Preserves abstract feature detection by convolutional layers
- For discrete classification, **relative order of outputs is much more important than absolute magnitudes**

Our Design

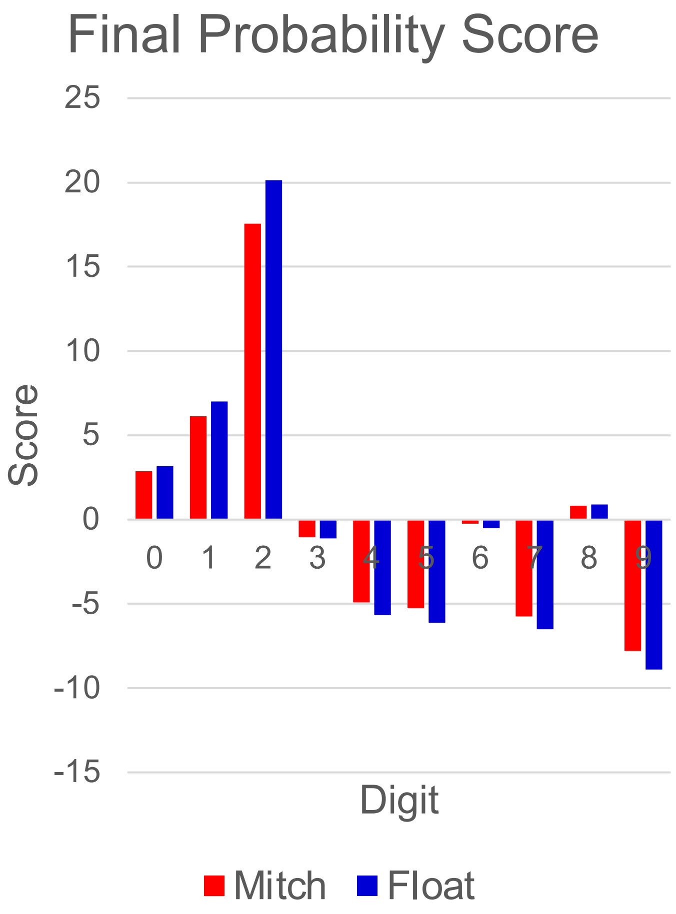


	1	2	3	4	5	6	7	8
1	-1.492	-0.2804	-0.1885	-1.06	-1.992	-2.17	-1.951	-1.545
2	-0.4241	-0.515	-1.045	-1.399	-0.5818	0.4006	0.211	-0.4155
3	-2.491	-1.7	-0.4084	-0.2086	1.035	0.9173	-0.0622	-0.7929
4	-2.475	-0.1783	0.9369	0.6956	-0.8547	-1.816	-1.343	-1.555
5	1.086	2.533	0.5643	-1.34	-1.914	-1.029	-0.291	-1.153
6	0.8449	0.8043	-1.277	-1.171	-0.5216	-0.3159	-0.2051	-1.429
7	-2.267	-1.161	-0.7545	0.2462	0.6309	-0.537	-0.8387	-1.831
8	-1.205	-0.2465	0.4825	0.7545	0.3141	-0.7325	-0.5985	-1.592

Floating-point

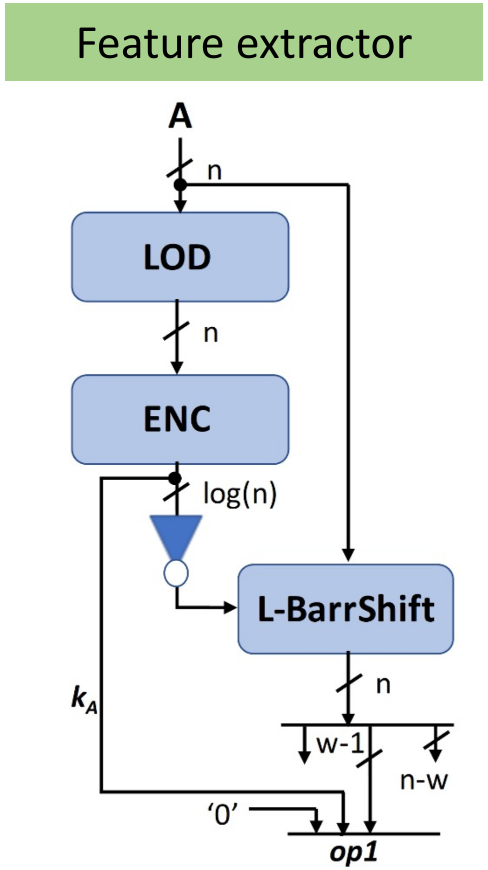


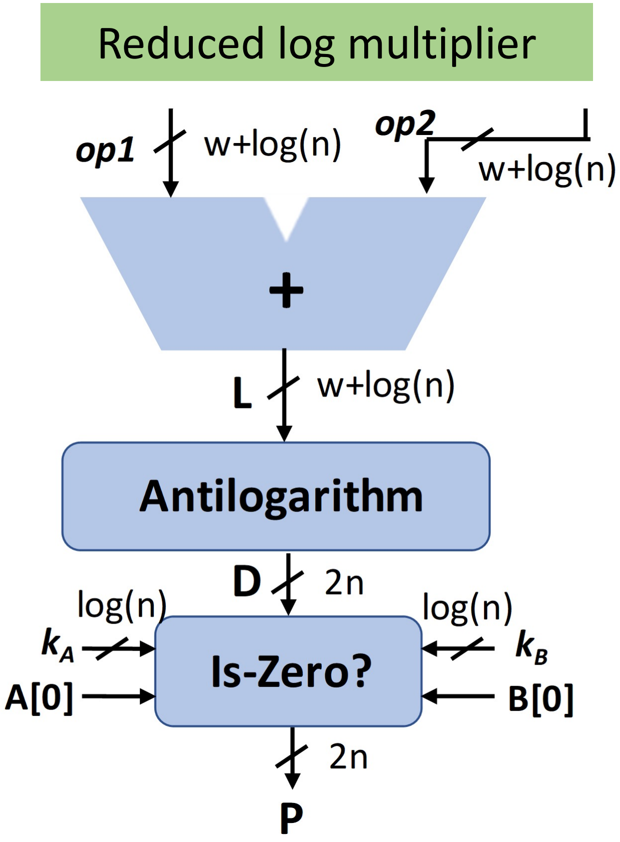
	1	2	3	4	5	6	7	8
1	-1.583	-0.2886	-0.2188	-1.136	-2.128	-2.28	-2.069	-1.62
2	-0.4435	-0.5412	-1.09	-1.517	-0.6648	0.4403	0.2043	-0.4493
3	-2.591	-1.842	-0.5007	-0.2205	1.129	0.9604	-0.09832	-0.8191
4	-2.664	-0.2438	1.027	0.7806	-0.8408	-1.883	-1.403	-1.608
5	1.144	2.744	0.683	-1.385	-2.014	-1.084	-0.2998	-1.173
6	0.9482	0.8823	-1.336	-1.254	-0.5176	-0.324	-0.2174	-1.493
7	-2.409	-1.264	-0.8253	0.2409	0.6382	-0.5483	-0.8777	-1.951
8	-1.288	-0.2705	0.5156	0.8044	0.324	-0.8071	-0.6587	-1.711

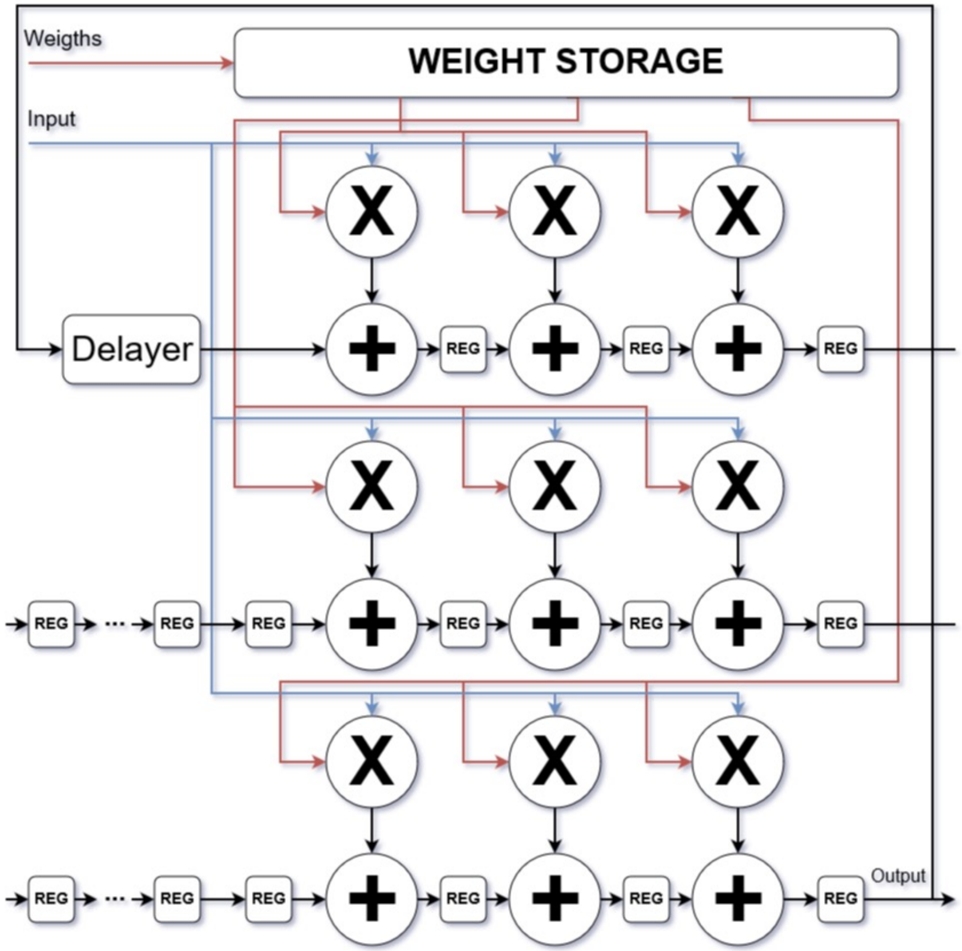


Approximate Log Multiplier: in an FPGA

- Store the results of the feature extractor (constant) and share to reduce the multiplier itself

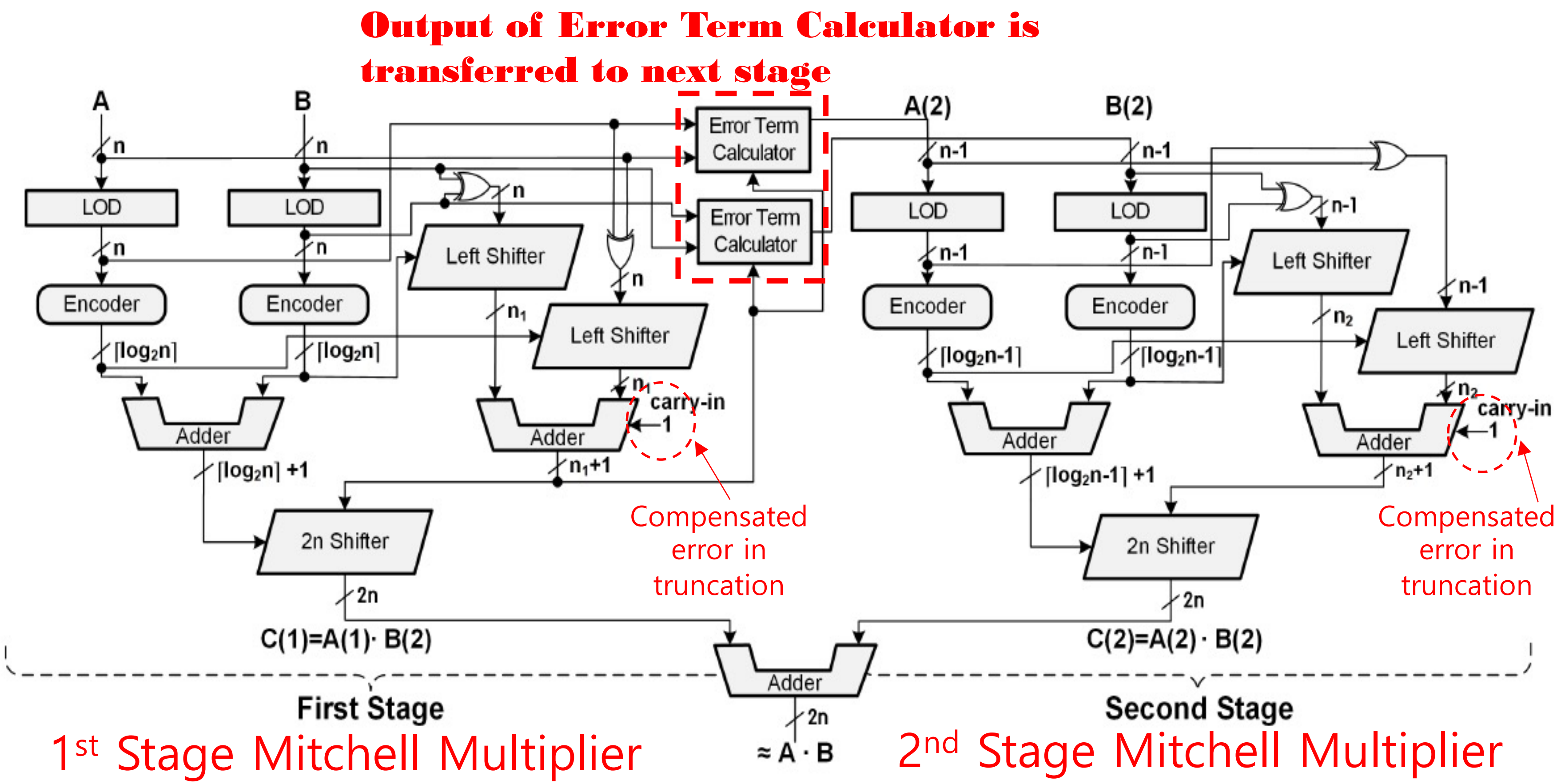






L. T. Oliveira, M. S. Kim, A. A. Del Barrio, N. Bagherzadeh and R. Menotti, "Design of Power-Efficient FPGA Convolutional Cores with Approximate Log Multiplier," in European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN'19), just accepted

Approximate Log Multiplier: iterative



- Basically we customize the bitwidth of every stage

Hyun-Jin Kim, Min Soo Kim, Alberto A. Del Barrio, Nader Bagherzadeh: A Cost-Efficient Iterative Truncated Logarithmic Multiplication for Convolutional Neural Networks. ARITH 2019: 108-111

Approximate Log Multiplier: iterative

Model	Dataset	Multiplier	Top-1 (%)	Top-5 (%)
NiN [11]	CIFAR-10	FLOAT[a]	89.4	-
		FIXED[b]	89.4	-
		MM[c]	88.7	-
		IM[d]	89.4	-
		PROP[e]	89.5	-
AlexNet [12]	ImageNet	FLOAT[a]	57.0	81.3
		FIXED[b]	57.0	81.3
		MM[c]	56.8	80.8
		IM[d]	56.8	81.3
		PROP[e]	56.9	81.3
GoogLeNet [13]	ImageNet	FLOAT[a]	68.3	88.4
		FIXED[b]	68.3	88.4
		MM[d]	67.1	87.5
		IM[d]	68.3	88.2
		PROP[e]	68.3	88.3
ResNet-50 [14]	ImageNet	FLOAT[a]	74.3	90.9
		FIXED[b]	74.2	90.9
		MM[c]	72.4	90.0
		IM[d]	73.9	90.9
		PROP[e]	73.8	90.6

[a] Original Caffe using floating-point multiplications
[b] Fixed-point multiplications
[c] Mitchell multipliers [2]
[d] Two-stage Babic's iterative multipliers [7]
[e] Proposed two-stage multiplier with $n_1 = 6, n_2 = 2$

- We tackle larger networks with high accuracy

n	design	$rerr_{max}$ (%)	$rerr_{avg}$ (%)	critical path (ns)	area (um^2)	power (uW)
8	Booth[a]	0	0	1.3	613	403
	MM[b]	11.11	3.76	1.3	446	217
	IM[c]	6.25	0.83	1.9	1,133	590
	PROP [d]	11.11	-1.09	2.6	786	370
16	Booth[a]	0	0	2.8	2,507	1,760
	MM[b]	11.11	3.85	2.3	1,168	602
	IM[c]	6.25	0.99	3.7	2,901	1,410
	PROP [e]	11.11	0.11	5.1	1,638	739
32	Booth[a]	0	0	5.4	10,139	6,750
	MM[b]	11.11	3.85	4.2	3,418	1,640
	IM[c]	6.25	0.99	6.5	7,674	3,680
	PROP [e]	11.11	0.12	7.9	3,102	1,370

[a] Radix-4 Booth multiplier
[b] Mitchell multiplier [2]
[c] Two-stage Babic's iterative multiplier [7]
[d] Proposed two-stage multiplier with $n_1 = 4, n_2 = 2$
[e] Proposed two-stage multiplier with $n_1 = 6, n_2 = 2$

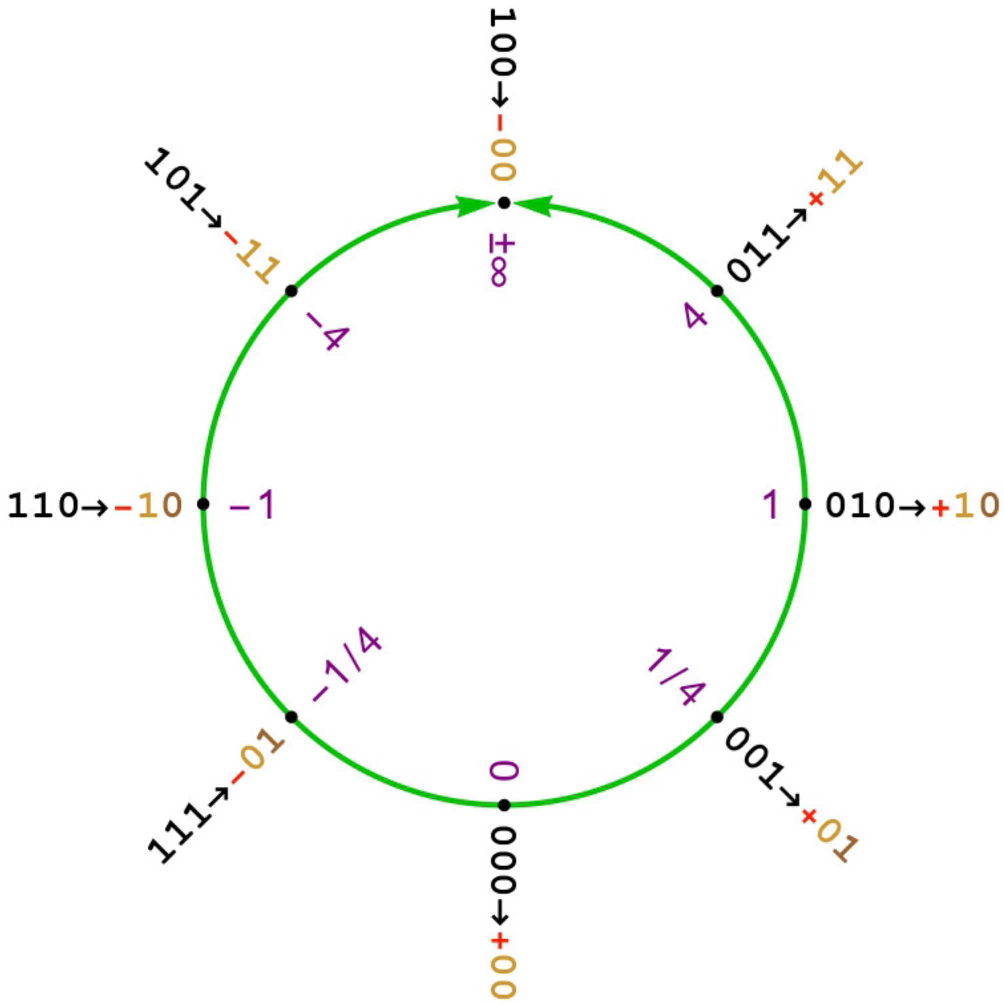
Hyun-Jin Kim, Min Soo Kim, Alberto A. Del Barrio, Nader Bagherzadeh: A Cost-Efficient Iterative Truncated Logarithmic Multiplication for Convolutional Neural Networks. ARITH 2019: 108-111

Outline

- Deep Learning and Approximate Computing
- Approximate Logarithmic Multiplication
- **The Posit Number System**
- Conclusions
- Open challenges

The Posit Number System (aka *unum v3*)

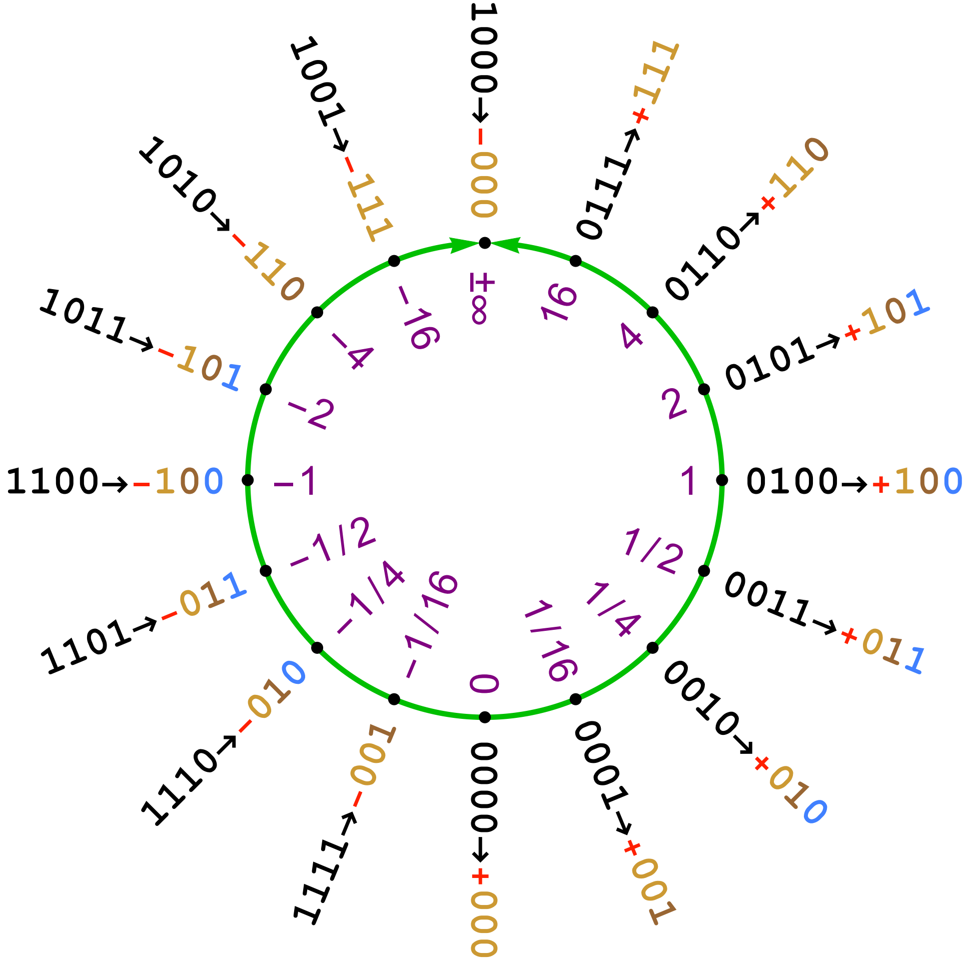
- Proposed by John L. Gustafson in 2017 as a direct drop-in replacement for floating-point numbers (IEEE 754)
- Better dynamic range
- No wasted patterns for denormal numbers
- Consistency between machines
  - Posit operations not rounded until the very end

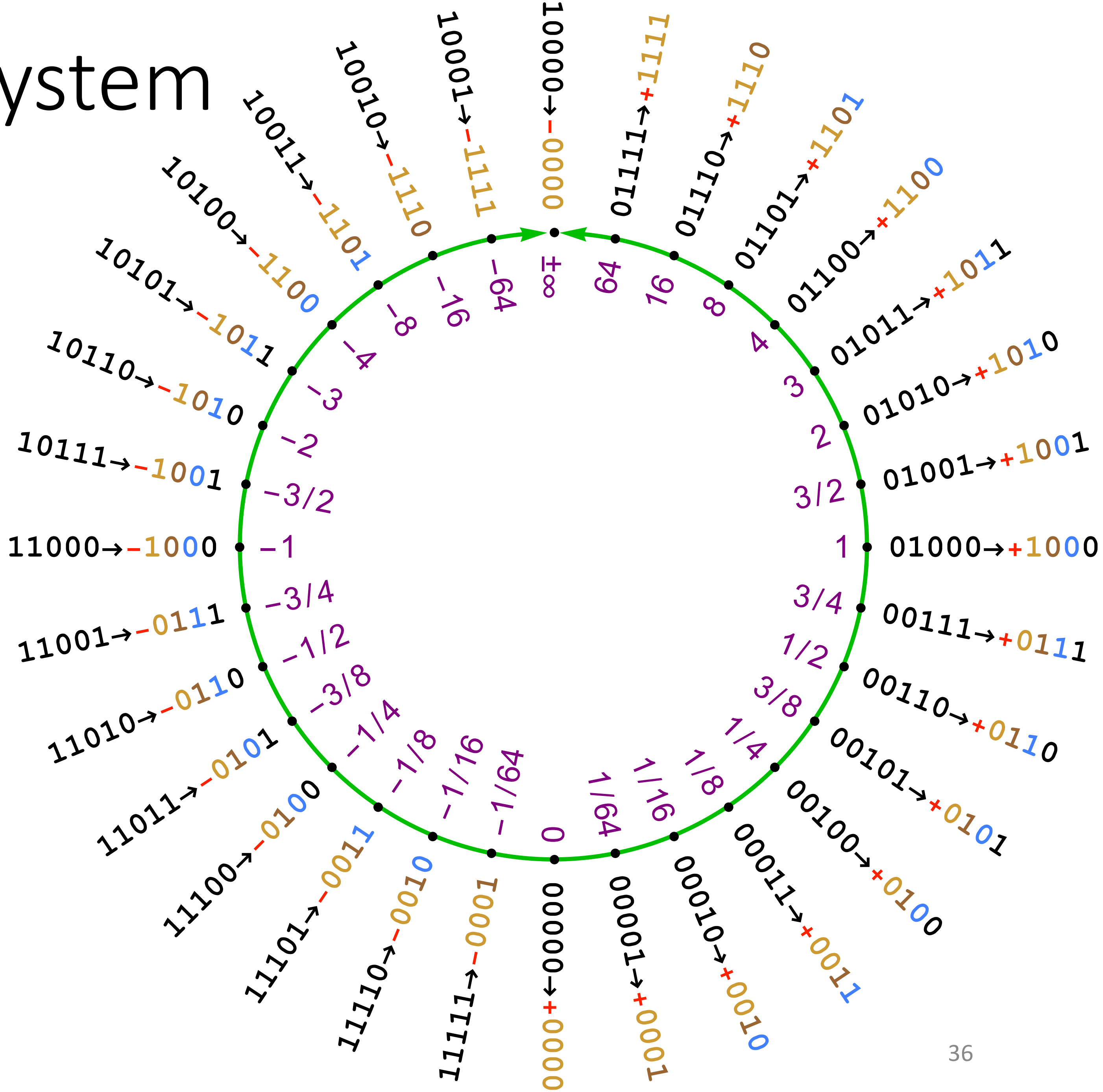


J. L. Gustafson and I. T. Yonemoto, “Beating floating point at its own game: Posit arithmetic,” *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, 06 2017.

The Posit Number System

- The order is a beauty in itself



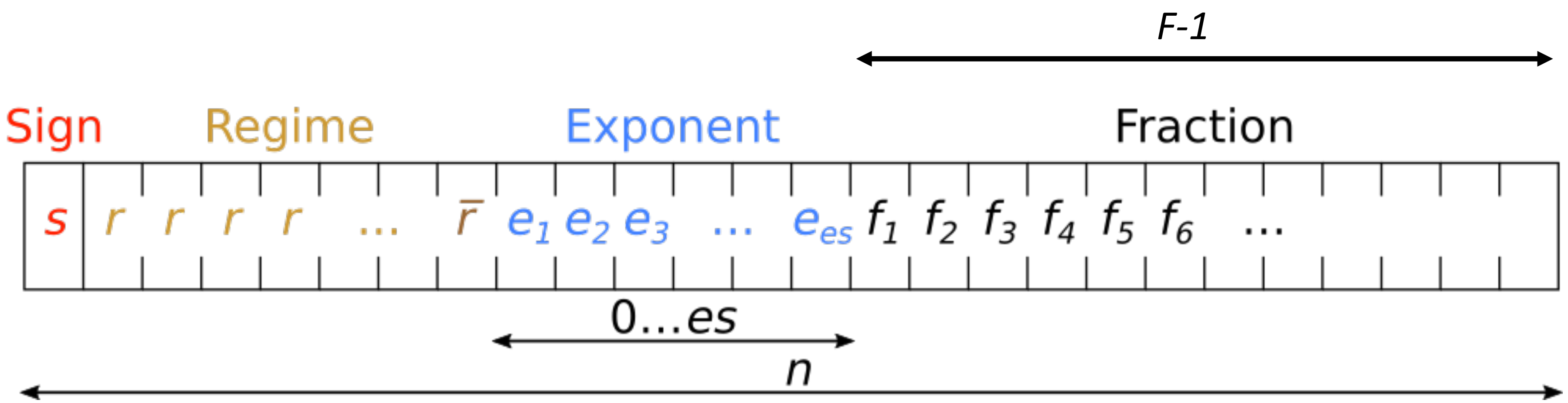


Numerical value of Posits

$$X = (-1)^s \times (useed)^k \times 2^e \times 1.f$$

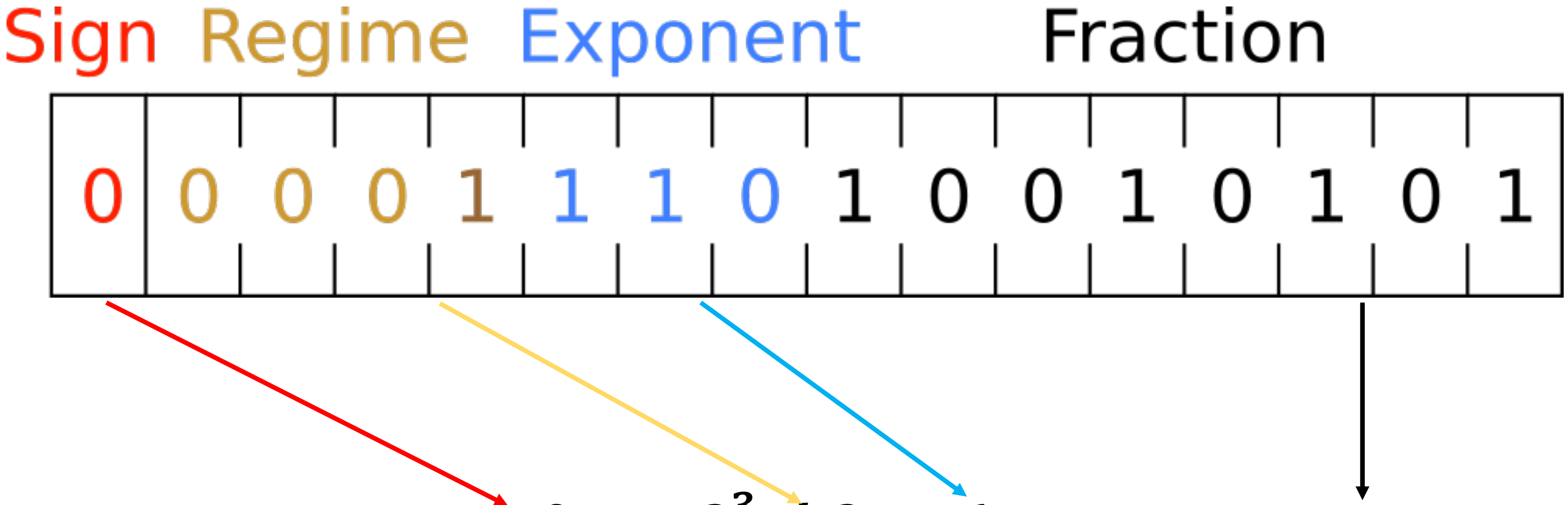
- s – sign
- $useed = 2^{2^{es}}$
- es – exponent size
- k – regime encoded value (signed integer)
- e – exponent value
- f – fraction value

Posit format encoding



- Sign bit (s)
- Regime (k) – sequence of r identical bits
  - #r = occurrences of r
  - $k = -\#r$ if r is 0, and $k = \#r - 1$ if r is 1
- Exponent (e) – represented by es bits
- Fraction (f) – unsigned integer divided by 2^F

Example: Posit⟨16,3⟩

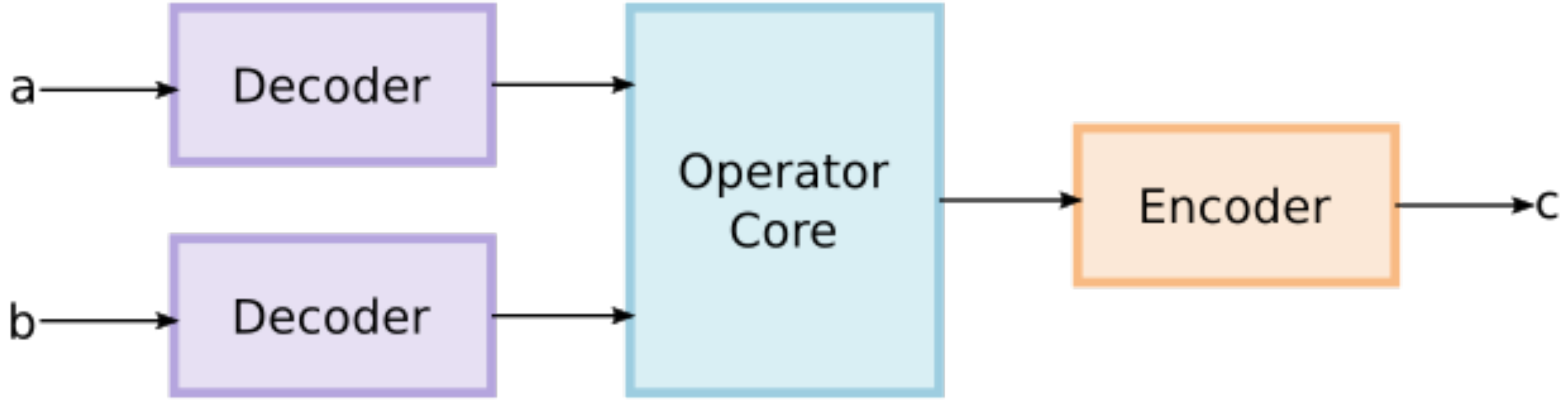


$$X = (-1)^0 \times (2^{2^3})^{-3} \times 2^6 \times (1 + 149/256)$$

$$= 6.034970283508301 \times 10^{-6}$$

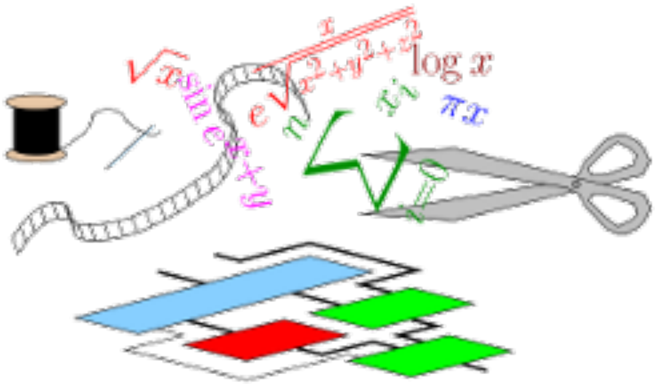
Posit functional units

- Posits were designed to be “hardware friendly”
  - Similar circuitry to floating point
  - Less special cases (just 0 and NaR)

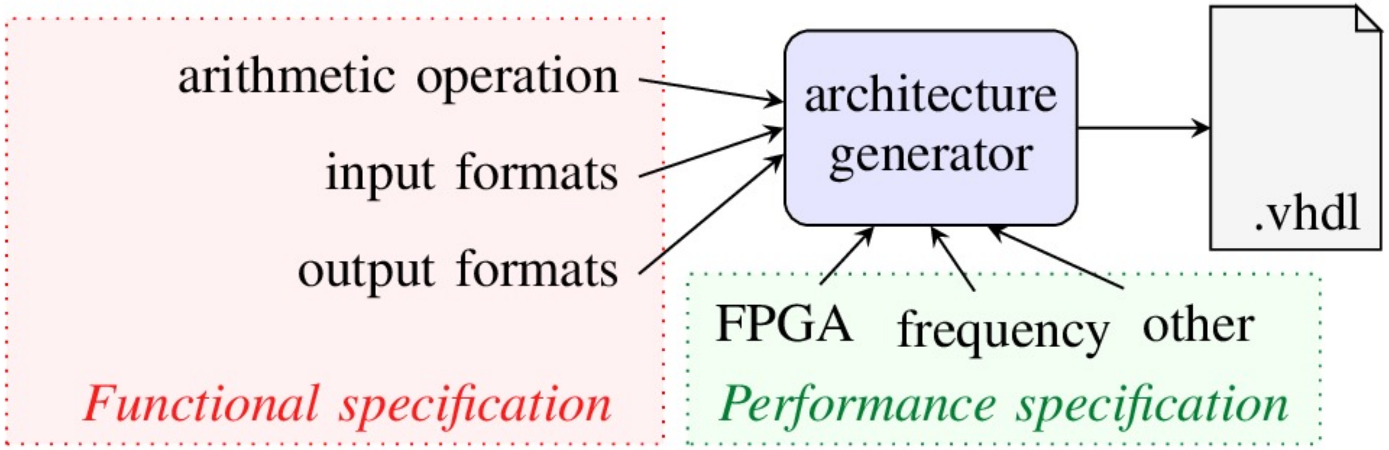


- Design challenge: runtime varying fields

FloPoCo Core Generator



- Open source tool for generating arithmetic cores for FPGA
- Operators are fully parameterized
- Written in C++, outputs synthesizable VHDL



Evaluation of Posit units - Adder

Still, not competitive w.r.t. floating point operators

	Posit $\langle n, es \rangle$	Area (μm^2)	Delay (ns)	Power (μW)	Energy (pJ)
PACoGen [1]	$\langle 16,1 \rangle$	3228.48	5.34	1637.6	8.74
	$\langle 32,2 \rangle$	7615.08	7.94	3828.3	30.4
Proposed	$\langle 8,0 \rangle$	1038.6	3.9	489.5	1.91
	$\langle 16,1 \rangle$	2176.92	6.23	1133.1	7.06
	$\langle 32,2 \rangle$	4880.88	9.48	2811.1	26.65
		-35.9%	+19.4%	**-30.8%**	**-19.2%**

[1] M. K. Jaiswal and H. K. -. So, "PACoGen: A Hardware Posit Arithmetic Core Generator," in IEEE Access, vol. 7, pp. 74586-74601, 2019, doi: 10.1109/ACCESS.2019.2920936.

R. Murillo, A. A. Del Barrio and G. Botella, "Customized Posit Adders and Multipliers using the FloPoCo Core Generator," *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, Sevilla, 2020, pp. 1-5, doi: 10.1109/ISCAS45731.2020.9180771.

Evaluation of Posit units - Multiplier

Still, not competitive w.r.t. floating point operators

	Posit $\langle n, es \rangle$	Area (μm^2)	Delay (ns)	Power (μW)	Energy (pJ)
PACoGen [1]	$\langle 16,1 \rangle$	4955.76	5.15	3036.6	15.64
	$\langle 32,2 \rangle$	15106.32	8.54	13027	111.25
Proposed	$\langle 8,0 \rangle$	1032.48	2.98	558.4	1.66
	$\langle 16,1 \rangle$	3321.72	5.64	2470.9	13.94
	$\langle 32,2 \rangle$	11924.64	8.87	11926	105.78
		-32.97%	+9.5%	**-18.6%**	**-10.86%**

[1] M. K. Jaiswal and H. K. -. So, "PACoGen: A Hardware Posit Arithmetic Core Generator," in IEEE Access, vol. 7, pp. 74586-74601, 2019, doi: 10.1109/ACCESS.2019.2920936.

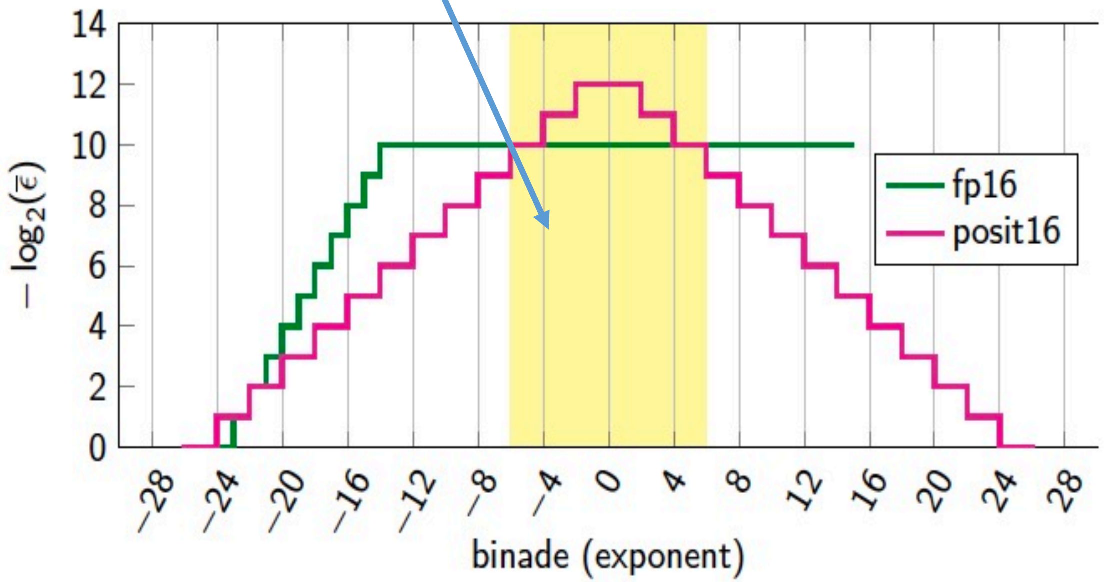
R. Murillo, A. A. Del Barrio and G. Botella, "Customized Posit Adders and Multipliers using the FloPoCo Core Generator," *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, Sevilla, 2020, pp. 1-5, doi: 10.1109/ISCAS45731.2020.9180771.

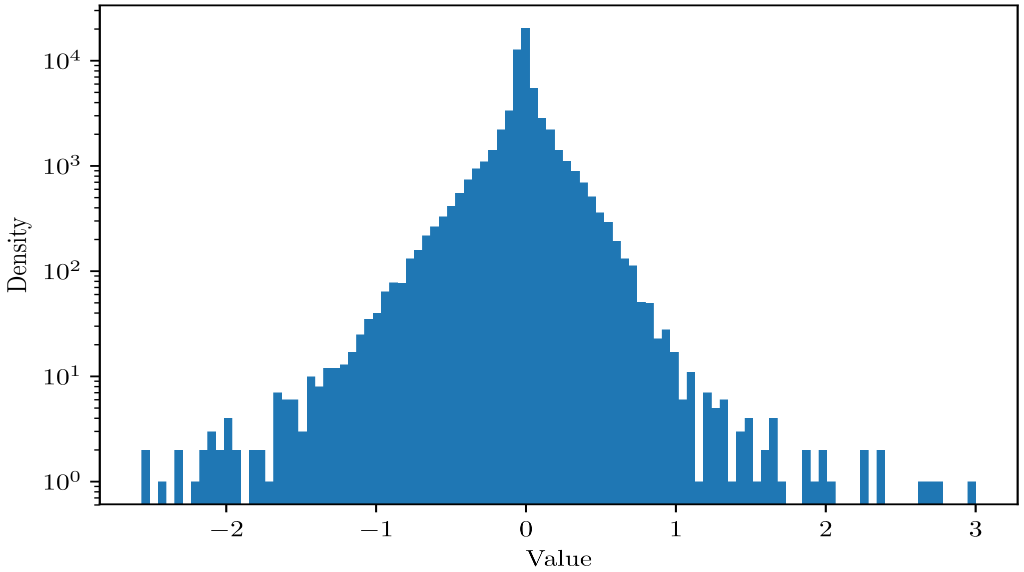
Why are posits interesting then?

- Posit format (J.L. Gustafson, 2017)

Tappered precision suits a gaussian distribution, i.e. like the weights of a DNN

Expectable: an *n/2* bits posit achieves the same accuracy as an n bits float

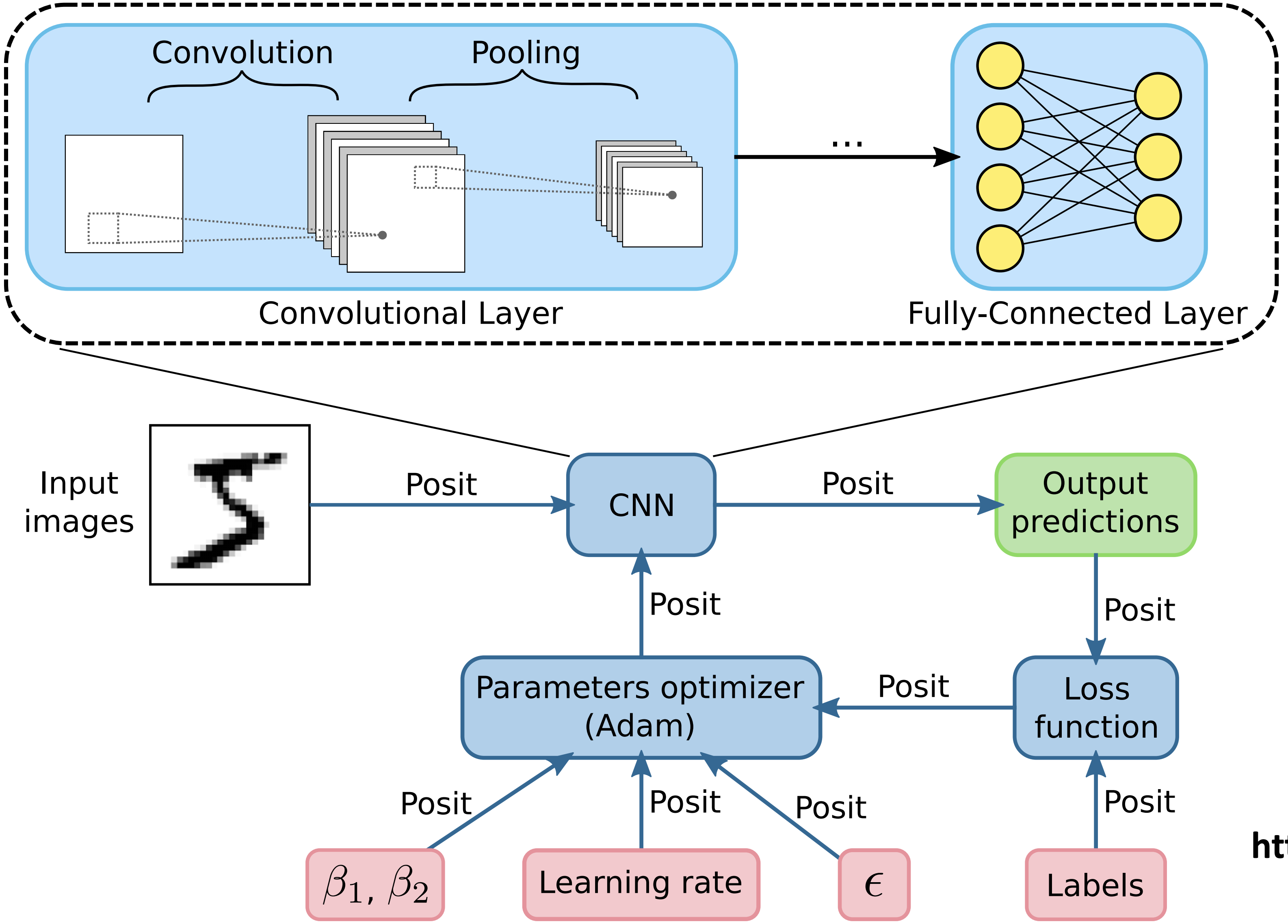




J. L. Gustafson and I. T. Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, 06 2017.

Deep PeNSieve

Raul Murillo, Alberto A. Del Barrio, Guillermo Botella: Deep PeNSieve: A deep learning framework based on the posit number system. Digit. Signal Process. 102: 102762 (2020)



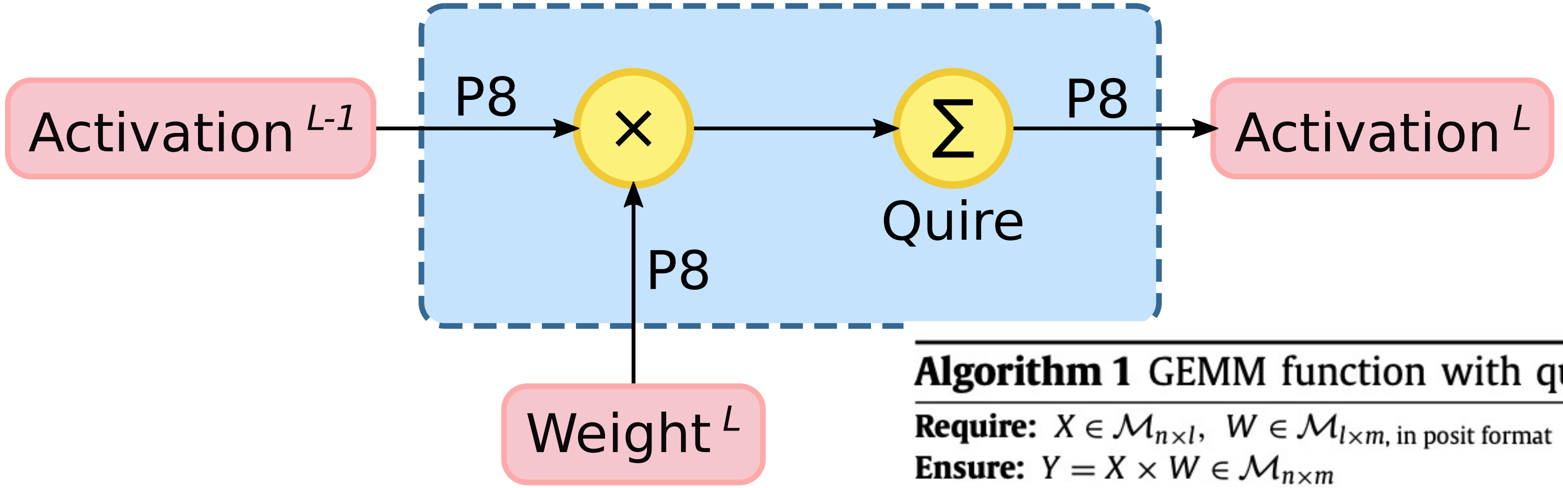
- Open-source framework based on TF
- Entire training performed with posits
  - Without conversions
- Allows training/inference with <32,2>, <16,1> and <8,0>

https://github.com/RaulMurillo/deep-pensieve

Deep PeNSieve

- Operations between 8-bit posits require a 64-bit quire (architectural register)
- This is where we round

In floating point format, the standard mandates rounding every operation. And every machine may have different rounding modes



Algorithm 1 GEMM function with quire.

Require: $X \in \mathcal{M}_{n\times l}$, $W \in \mathcal{M}_{l\times m}$, in posit format
Ensure: $Y = X \times W \in \mathcal{M}_{n\times m}$

for all $i \in [0, n]$ **do**
  for all $j \in [0, m]$ **do**
    $q \leftarrow quire(0)$
    for all $k \in [0, l]$ **do**
      $q \leftarrow x_{i,k} \cdot w_{k,j} + q$ //The result is accumulated, but not rounded
    end for
    $y_{i,j} \leftarrow posit(q)$ //The operation is rounded here
  end for
end for

Raul Murillo, Alberto A. Del Barrio, Guillermo Botella: Deep PeNSieve: A deep learning framework based on the posit number system. Digit. Signal Process. 102: 102762 (2020)

Deep PeNSieve

Quite remarkable: even higher precision than float

Table 1
Accuracy results for the inference stage.

Format	MNIST		Fashion-MNIST		SVHN		CIFAR-10	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Float 32	99.17%	100%	89.34%	99.78%	89.32%	98.35%	68.06%	95.15%
Posit⟨32, 2⟩	99.09%	99.98%	89.90%	99.84%	89.51%	98.36%	69.32%	96.59%
Posit⟨16, 1⟩	99.18%	100%	90.17%	99.81%	90.90%	98.72%	72.51%	97.40%

Table 2
Post-training quantization accuracy results for the inference stage.

Format	MNIST		Fashion-MNIST		SVHN		CIFAR-10	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Float 16	99.17%	100%	89.34%	99.78%	89.32%	98.35%	68.05%	96.15%
INT8	99.16%	100%	89.51%	99.79%	89.33%	98.38%	68.15%	96.14%
Posit⟨8, 0⟩	98.77%	99.99%	88.52%	99.82%	81.31%	97.07%	43.89%	86.49%
$\text{Posit}\langle 8, 0\rangle_{\text{quire}}$	99.07%	99.99%	89.92%	99.81%	89.13%	98.39%	68.88%	96.47%

Raul Murillo, Alberto A. Del Barrio, Guillermo Botella: Deep PeNSieve: A deep learning framework based on the posit number system. Digit. Signal Process. 102: 102762 (2020)

Outline

- Deep Learning and Approximate Computing
- Approximate Logarithmic Multiplication
- The Posit Number System
- **Conclusions**
- Open challenges

Conclusions

- ML and DNNs have opened new possibilities to Computer Arithmetic
- Approximate Computing suits the error tolerance of these applications
- Good Enough Arithmetic is critical to find the best tradeoff
    - Accuracy vs Energy
    - There is no need to be better
- New Generation Arithmetic (NGA) is here
    - Energy efficient
    - Even with better features

Outline

- Deep Learning and Approximate Computing
- Approximate Logarithmic Multiplication
- The Posit Number System
- Conclusions
- <u>**Open challenges**</u>

Open challenges

- Integrating logarithmic arithmetic in an accelerator
  - Memory accesses and other details must be considered too
  - High-level Synthesis is not to be forgotten, can enhance the arithmetic approach [1]
- Posit units are still not competitive with respect to IEEE-754 based or bfloat16
  - Posits are not standard yet
  - The community is still understanding the properties of the new format
  - New tricks are required

[1] A. A. Del Barrio, R. Hermida and S. Ogrenci-Memik, "A Combined Arithmetic-High-Level Synthesis Solution to Deploy Partial Carry-Save Radix-8 Booth Multipliers in Datapaths," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 66, no. 2, pp. 742-755, Feb. 2019. doi: 10.1109/TCSI.2018.2866172

Open challenges

- Training with posits is very slow, every operation must be emulated
  - 10 days with CIFAR-10
  - The framework can be optimized yet (SW)
  - RISC-V processor can integrate posit support (HW and SW)
  - https://www.redleonardo.es/beneficiario/alberto-antonio-del-barrio-garcia/

This work was produced with the support of a 2020 Leonardo Grant for Researchers and Cultural Creators, BBVA Foundation, whose id is PR200320/01

Fundación BBVA

THANKS SO MUCH FOR YOUR ATTENTION !!

Any questions ??? … or you can email me at abarriog@ucm.es