Rethinking Floating Point for Deep Learning

Floating Point (IEEE 754 2008)

- B: radix, 2 orr 10
- p: # of digits in significant (precision)
- emax: max exponent
- emin: min exponent = 1 – emax

Binary Format

Parameter	binary16	**binary32**	**binary64**	binary128	binary{k} ($k \geq 128$)
k	16	**32**	**64**	128	multiple of 32 $1 + w + t$
p	11	**24**	**53**	113	$k - round(4 \times \log_2 k) + 13$
emax	15	**127**	**1023**	16383	$2^{k-p-1} - 1$
bias, $E - e$	15	**127**	**1023**	16383	$emax$
sign bit	1	1	1	1	1
w	5	**8**	**11**	15	$round(4 \times \log_2 k) - 13$
t	10	**23**	**52**	112	$k - w - 1$

- k : storage width in bits. 저장 비트 수.
- p : precision in bits. the number of digits int the significant(precision).
- emax : the maximum exponent e
- sign bit : 부호 표시. 0 이면 $+$이고, 1 이면 $-$ 이다.
- w : exponent field width in bits. 지수부 비트 수.
- t : trailing significand field width in bits. 가수부 비트 수.

Floating Point Representation

- Signed zeros
- $(-1)^s$ x b^e x m
- Two infinities (positive and negative)
- Two NaNs, qNaN and sNaN
- Denormals

Efforts for reducing computational complexity

- Fixed point
- Uniform quantization via 8 bit integer
- Ternary representation
- Binary/low-bit representation
- etc

Floating Point Variants

- (e, s) float (IEEE 754 type)
- Blocking floating point
- LNS
- Posit
- Tapered floating point
  - exponent & significand size varies
- Posit by Gustafson

(N,s)-Posit (Gustafson)

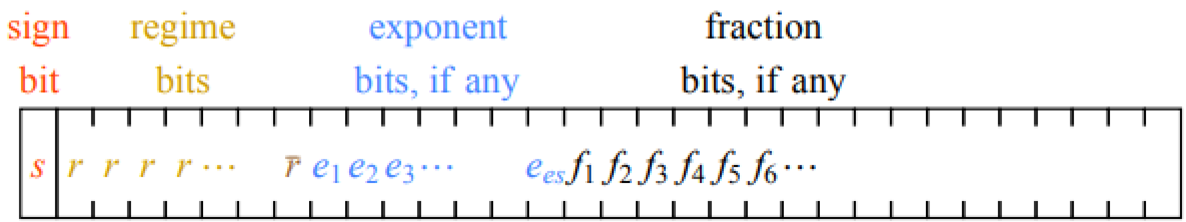

Figure 2. Generic posit format for finite, nonzero values

As an example, fig. 4 shows a build up from a 3-bit to a 5-bit posit with $es = 2$, so $useed = 16$:

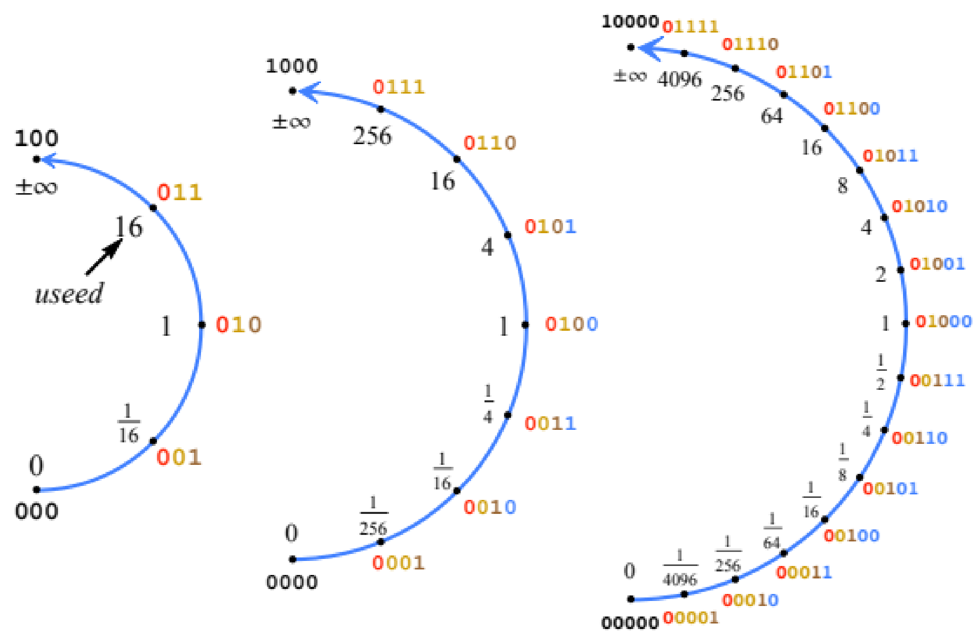

Figure 4. Posit construction with two exponent bits, $es = 2$, $useed = 2^{2^{es}} = 16$

Accumulator

- FP accumulation is not associative
- FMA(fused multiply add) is frequent
- c + ab = c ..... Problem
- Kulisch accumulator
  - fixed point register enough for $\pm(f_{max}^2 + f_{min}^2)$
  - shift and add
  - called EMA(exact multiply add)
  - $r(\sum_i a_i b_i)$ vs $r(a_n b_n + r(a_{n-1} b_{n-1} + r(\cdots + r(a_1 b_1 + 0) \cdots)))$
  - EMA can be more efficient

ELMA, (N,s,α,β,γ) log (exact log-linear multiply-add)

- Logarithmic number system avoids hardware multipliers
- σ is the weak point
  - costly LUT & linearization
- Translate m.f to linear domain

  m -> 2^m

  f -> g=p(f)=2^f–1 using $(2^{f_{bits}} \times \alpha)$-bit LUT.
- Kulisch-accumulated
- Back to log using $(2^{\beta} \times \gamma)$-bit LUT.
- if $\alpha \geq f_{bits}+1, \beta \geq \alpha, \gamma = f_{bits}$, $f = r(q(r(r(p(f),\alpha),\beta)),\gamma)$ is identity

$$i = \log_2(x), j = \log_2(y)$$
$$\log_2(x \pm y) = i + \sigma_{\pm}(j-i)$$
$$\log_2(xy) = i + j$$
$$\log_2(x/y) = i - j$$
$$\sigma_{\pm}(x) = \log_2(1 \pm 2^x)$$

FPGA experiments

Table 2: ResNet-50 ImageNet validation set accuracy per math type

Math type	Multiply-add type	top-1 acc (%)	top-5 acc (%)
float32	FMA	76.130	92.862
(8, 1, 5, 5, 7) log	**ELMA**	**-0.90**	**-0.20**
(7, 1) posit	EMA	-4.63	-2.28
(8, 0) posit	EMA	-76.03	-92.36
(8, 1) posit	EMA	-0.87	-0.19
(8, 2) posit	EMA	-2.20	-0.85
(9, 1) posit	EMA	-0.30	-0.09
Jacob et al. [15]:			
float32	FMA	76.400	n/a
int8/32	MAC	-1.50	n/a
Migacz [23]:			
float32	FMA	73.230	91.180
int8/32	MAC	-0.20	-0.03

Chip Area and Power

Table 3: Chip area and power for 28 nm, 1-cycle multiply-add at 500 MHz

Component	Area μm^2	Power μW
int8/32 MAC PE	**336.672**	**283**
multiply	121.212	108.0
add	117.810	62.3
non-combinational	96.768	112.7
(8, 1, 5, 5, 7) log ELMA PE	**376.110**	**272**
log multiply (9 bit adder)	32.760	17.1
$r(p(f))$ (16x5 bit LUT)	8.946	5.4
Kulisch shift ($6 \rightarrow 38$ bit)	81.774	71.0
Kulisch add (38 bit)	123.732	54.2
non-combinational	126.756	124.3
float16 (w/o denormals) FMA PE	**1545.012**	**1358**
(5, 10) (11, 11, 10) log ELMA PE	**1043.154**	**805**
(this log is (5, 10) float16-style encoding, same dynamic range; denormals for log and float16 here are unhandled and flush to zero)		
32x32 systolic w/ int8/32 MAC PEs	**348231**	**226000**
32x32 systolic w/ (8, 1, 5, 5, 7) log ELMA PEs	**457738**	**195500**