Study of Posit Numeric in Speech Recognition Neural Inference

Author: Zishen Wan, Eric Mibuari, En-Yu Yang, Thierry Tambe
Harvard University
Cambridge, MA

Advisor: 陳鵬升 教授
Presenter: 洪祐鈞

Intuition

- Posit: A new ways to represent floating-numbers
  - Good at represent the |number| close to 1
- Text-to-Speech model is RNN, attention based model.
- Inference: Given input to existing model and get the output.
- Quantize: Fit big data to small data.

How does Posit Works?

- Posit Format:

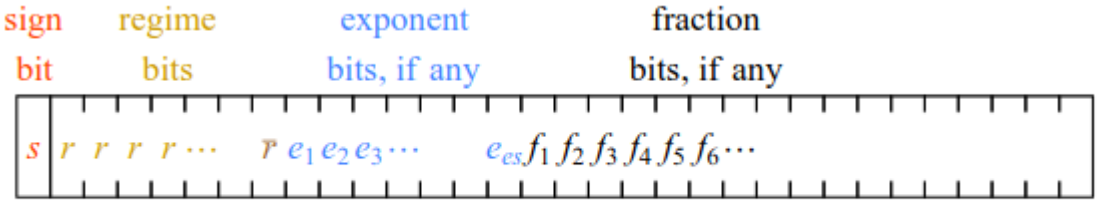


Figure 2. Generic posit format for finite, nonzero values

- Parameter:
  - Posit<total_bits, es_bits>
    - Example: Posit<32,3>
      - Total of 32-bit. 32-bit posit
      - es: 3
        - At most 3 bit of exponent
        - 3-bit of exponent as a pack

Why do I choose this paper?

- My research focus is on
  - Integrating posit floating point arithmetic in to MLIR infrastructure
    - MLIR: Framework of Compiler Framework
  - Specifically on deep learning application.
- Another research topic of lab is optimizing or quantize the transformer model, a lot of operator e.g attention layer similar to this.

How does Posit Works?

- Posit Format:

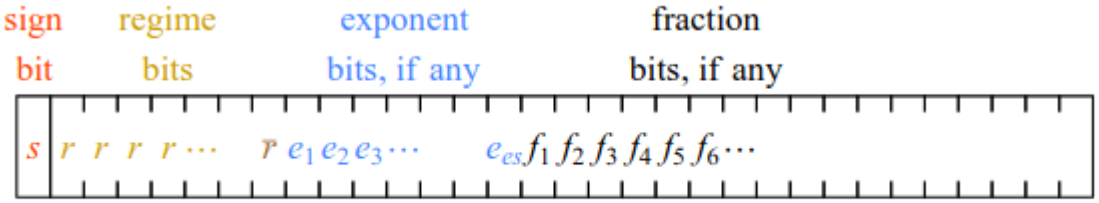


Figure 2. Generic posit format for finite, nonzero values

$$x = \begin{cases} 0, & p = 0, \\ \pm\infty, & p = -2^{n-1}, \\ sign(p) \times useed^k \times 2^e \times f, & \text{all other } p. \end{cases}$$

- 00000000: Zero
- 10000000: NaR (Not a Real)
- useed = $2^{2^{es}}$

How does Posit Works?

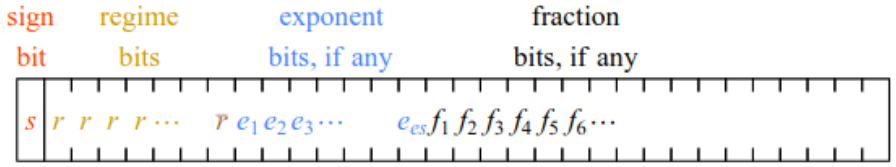


Figure 2. Generic posit format for finite, nonzero values

$$x = \begin{cases} 0, & p = 0, \\ \pm\infty, & p = -2^{n-1}, \\ sign(p) \times useed^{k} \times 2^{e} \times f, & \text{all other } p. \end{cases}$$

$$2^{2^{es}}$$

- sign term:
  - 0 -> +
  - 1 -> -
- exponent term:
  - 101 -> e = 5
  - 2^{e} , no Bias like IEEE754
  - at most es-bit
- fraction term:
  - 1011 -> 1 + (0.5 + 0.125 + 0.0625)
  - f = $1.f_1f_2f_3f_4$... like IEEE754,
  - but no subnormal number.

How does Posit Works?

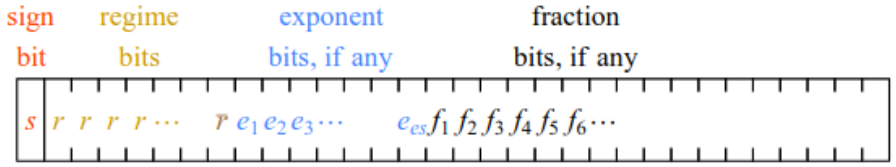


Figure 2. Generic posit format for finite, nonzero values

$$x = \begin{cases} 0, & p = 0, \\ \pm\infty, & p = -2^{n-1}, \\ sign(p) \times \boxed{useed^{k}} \times 2^{e} \times f, & \text{all other } p. \end{cases}$$

$2^{2^{es}}$

regime term:

- Find until the first opposite bit.
- 001 -> k = -2,
- 01 -> k = -1
- 10 -> 0,
- 110 -> k = 1,
- 1110 -> k = 2,
- k value intuitively treat as carry of pack of exponent
- Decide $\text{useed}^{k} = 2^{(2^{es})^{k}}$
- __builtin_clz() -> x86 instruction LZCNT

Insight on posit

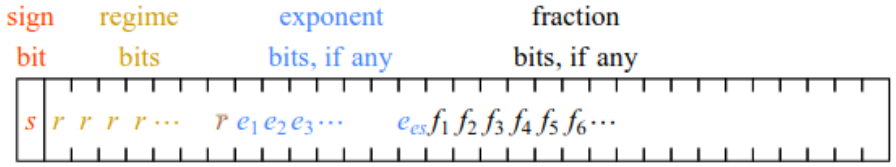


Figure 2. Generic posit format for finite, nonzero values

- "0000111111111111" next posit is:
  "0000010000000000"
- We need larger precision at smaller exponent.
  - We have more fraction bit at smaller exponent.
- Value close to exponent zero has shortest regime bits.
  - Good at represent the |number| close to 1

Convert from posit to real number:

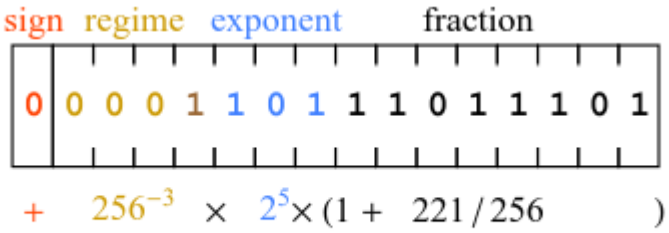


Figure 5. Example of a posit bit string and its mathematical meaning

- Posit<16, 3>: 16-bit, es = 3
- sign term: 0 -> positive -> +
- regime term: 0001 -> k = -3, -> 256^3
  - Useed = $2^{2^{es}}$
- exponent term: 101 = 5 -> 2^{-5}
- fraction term: 1+ (0.5 + 0.25 + 0.0625 + ...)

Convert from posit to real number:

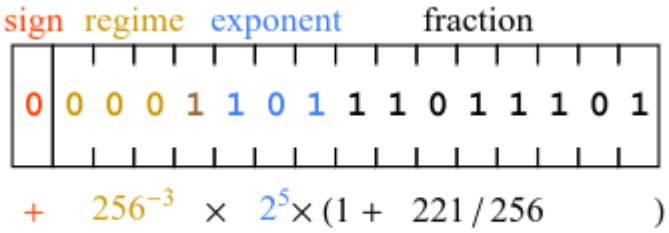


Figure 5. Example of a posit bit string and its mathematical meaning

- In Paper:
  - Add check (zero, NaR..) exception
  - Check regime bit length -> exp -> fraction
    - exp and fraction bit may not exist.
  - Multiply those sign, regime, exponent, fraction terms

Convert from real number to posit:

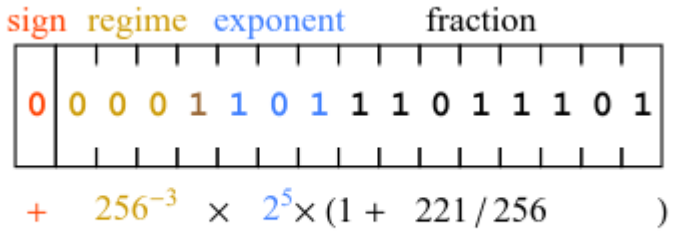


Figure 5. Example of a posit bit string and its mathematical meaning

- Convert 694.20 to posit<32,2>
  - Es = 2
  - Useed = $2^{(2^2)} = 2^4$
  - 694.20 = 2^9 * 1.359
  - 9/4 = 2 ... 1 -> k = 2, e = 1

Convert from real number to posit:

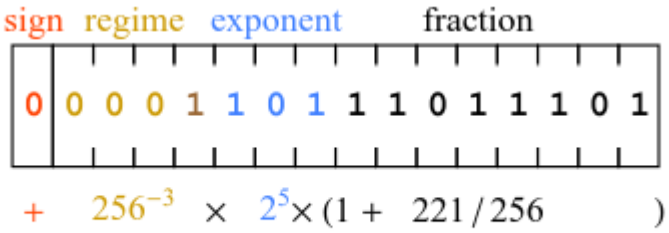


Figure 5. Example of a posit bit string and its mathematical meaning

- Convert 694.20 to posit<32,2>
  - sign bit = 0 (positive)
  - regime bit = 1110 (k = 2)
  - exponent bit = 01 (e = 1)
  - 1.359 = 1 + ($2^{-2} + 2^{-4} + 2^{-5} + 2^{-7} + 2^{-8}$ …)
  - fraction bit = 01011011000110011001100110011

Convert from real number to posit:

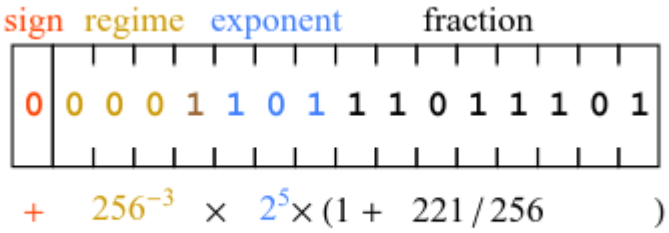


Figure 5. Example of a posit bit string and its mathematical meaning

- This paper has an error: (useed > |value| > 1)

In this case, we'll not really use the regime part of the bit word. However, we still need to use at least two bits for the regime, one to indicate that our regime exponent is zero, and the other to show the regime termination. That is, the scale factor coming out of the regime part becomes 1 (exponentiation by zero).

Instead of steps 1 and 2 of the algorithm above, we set the regime bits to "01". The real number is then used as the exponent and

Larger Dynamic

- Posit have larger dynamic range

Table 3. Float and posit dynamic ranges for the same number of bits

Size, Bits	IEEE Float Exp. Size	Approx. IEEE Float Dynamic Range	Posit *es* Value	Approx. Posit Dynamic Range
16	5	6×10^{-8} to 7×10^{4}	1	4×10^{-9} to 3×10^{8}
32	8	1×10^{-45} to 3×10^{38}	3	6×10^{-73} to 2×10^{72}
64	11	5×10^{-324} to 2×10^{308}	4	2×10^{-299} to 4×10^{298}
128	15	6×10^{-4966} to 1×10^{4932}	7	1×10^{-4855} to 1×10^{4855}
256	19	2×10^{-78984} to 2×10^{78913}	10	2×10^{-78297} to 5×10^{78296}

Accuracy

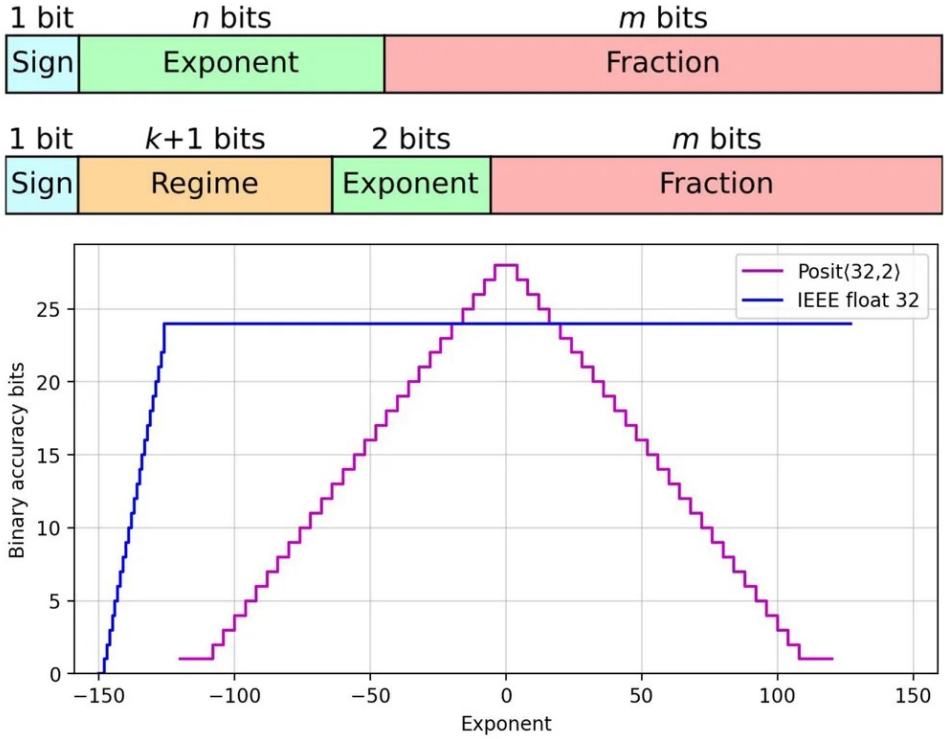
- IEEE FP32: 23-bit fraction + 1(1.xxx fraction)
- Posit 32: 32 - 1(sign) - 2(regime) - 2(exponent) + 1(1.xxx fraction)
- IEEE FP16: 10-bit fraction + (1.xxx fraction)
- Posit 16: 16 - 1(sign) - 2(regime) - 1(exponent) + 1(1.xxx fraction)

Table 2: Accuracy Compared with posit and float

Size, bit	Float Max Accuracy, bits	Posit Max Accuracy, bits	Range where accuracy: posit ≥ float
8	5	6	1/4 to 4
16	11	13	1/64 to 64
32	24	28	$1. \times 10^{-6}$ to $1. \times 10^{6}$

Posit(32, 2) accuracy v.s. IEEE float 32

- Posit has more accuracy when the value's 2's exponent is close to 0



(ref: https://spectrum.ieee.org/floating-point-numbers-posits-processor)

Posit Precision

- Observation:
  - Shorter regime bits allows more fraction bits
  - Value distribution is close together exponentially around 0.

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

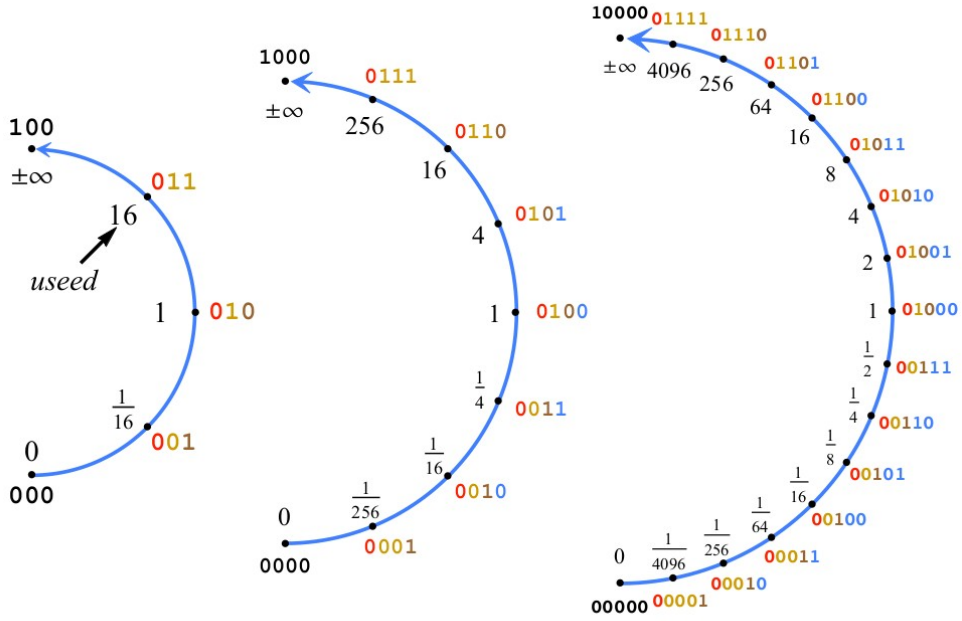


Figure 4. Posit construction with two exponent bits, $es = 2$, $useed = 2^{2^{es}} = 16$

Paper Abstract

- RNN model is huge for memory aspect.
- Quantize the mode is good for deploy to edge device.
- Research various kind of floating point format on speech-to-text model inference performance
- Result shows posit<8,0> is the best for aggressive quantization.
- Posit hardware is more efficient then fixed-point based, in terms of area and power cost.

Contribution

- Develop a Python-based framework for converting between float, fixed-point and posit numbers.
- Compare and evaluate the posit and various data types in neural speech recognition inference.
- Design hardware unit of posit, fixed point and floating point.
- Overall hardware prototype of speed-to-text inference.

- Sad, it seems like it's not open-source

Overview of common numerical data type

- In Paper comparison:
- BF16: Used in Google cloud TPU and Intel AI processors.
- Fixed-point: 0.1 + 0.2 != 0.30000000000000004
- Low precision fixed-point arithmetic become DL quantization in inference.

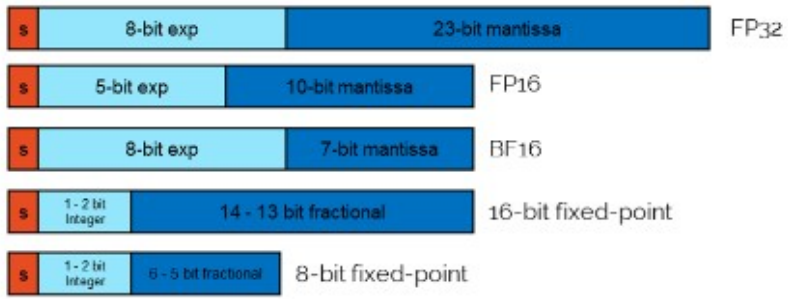


Figure 2: Common numerical data types used in Today's computers and ASICs

Overview of common numerical data type

- In Pa
- BF16: U
- Fixed-p
- Low pre

Other FP Format

Nvidia、Arm與Intel公布FP8規格，企圖成為AI交換格式標準

這三家科技大廠發布《供深度學習使用的FP8格式》（FP8 Formats for Deep Learning）白皮書，用以描述8位元浮點規格，提供一個共同的格式來加速包括AI訓練與推理的AI發展

文/ 陳曉莉 | 2022-09-16 發表

讚 147 分享

背景圖片來源／Nvidia

iT+ 看影片

金融業
中應用
策略
Cloud Sum
26 分

資料庫
綜觀全
勢 - 金
析
CipherTech

識別出
smell
MWC | 34

Other FP Format

FP8 Formats For Deep Learning

Paulius Micikevicius, Dusan Stosic, Patrick Judd, John Kamalu, Stuart Oberman, Mohammad Shoeybi, Michael Siu, Hao Wu
NVIDIA
{pauliusm, dstosic, pjudd, jkamalu, soberman, mshoeybi, msiu, skyw}@nvidia.com

Neil Burgess, Sangwon Ha, Richard Grisenthwaite
Arm
{neil.burgess, sangwon.ha, richard.grisenthwaite}@arm.com

Naveen Mellempudi, Marius Cornea, Alexander Heinecke, Pradeep Dubey
Intel
{naveen.k.mellempudi, marius.cornea, alexander.heinecke, pradeep.dubey}@intel.com

Abstract

FP8 is a natural progression for accelerating deep learning training inference beyond the 16-bit formats common in modern processors. In this paper we propose an 8-bit floating point (FP8) binary interchange format consisting of two encodings - E4M3 (4-bit exponent and 3-bit mantissa) and E5M2 (5-bit exponent and 2-bit mantissa). While E5M2 follows IEEE 754 conventions for representatio of special values, E4M3's dynamic range is extended by not representing infinities and having only one mantissa bit-pattern for NaNs. We demonstrate the efficacy of the FP8 format on a variety of image and language tasks, effectively matching the result quality achieved by 16-bit training sessions. Our study covers the main modern neural network architectures - CNNs, RNNs, and Transformer-based models, leaving all the hyperparameters unchanged from the 16-bit baseline training sessions. Our training experiments include large, up to 175B parameter, language models. We also examine FP8 post-training-quantization of language models trained using 16-bit formats that resisted fixed point int8 quantization.

Other FP Format

EE Times

DESIGNLINES | AI & BIG DATA DESIGNLINE

Nvidia's Blackwell Offers FP4, Second-Gen Transformer Engine

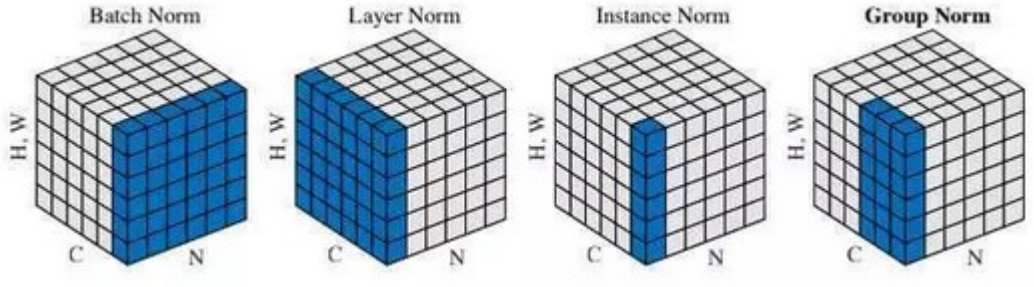
By Sally Ward-Foxton 03.27.2024 0

SAN JOSE, CALIF.—Market leader Nvidia unveiled its new generation of GPU technology, designed to accelerate training and inference of generative AI. The new technology platform is named Blackwell, after game theorist David Harold Blackwell, and will replace the previous generation, Hopper.

"Clearly, AI has hit the point where every application in the industry can benefit by applying generative AI to augment how we make PowerPoints, write documents, understand our data and ask questions of it," Ian Buck, VP and general manager of Nvidia's hyperscale and HPC computing business, told EE Times. "It's such an incredibly valuable tool that the world can't build up infrastructure fast enough to meet the promise, and make it accessible, affordable and ubiquitous."

Experiment

- Train 2 speech recognition model
  - Framework: OpenNMT toolkit
  - Dataset: LibriSpeech corpus:
  - 1,000 hours of audiobooks
  - Model architecture follows DeepSpeech3 specifications
  - Substitute batch normalization to layer normalization.



Experiment

- Train 2 speech recognition model
  - Networks were not retrained after quantization was applied
  - There's no specific words describe how the quantization is done
    - From the context, the conversion is based on real value.

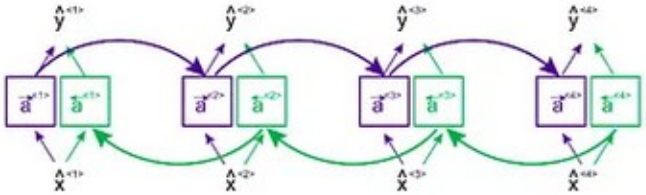
Experiment

- Train 2 speech recognition model
  - Model 1:
    - MLP attention model
  - Encoder:
    - 4 layer of bidirectional-GRU (Gated Recurrent Unit)
      - 1024 hidden units
    - 2 downsampling pooling layer
  - Decoder: 1 forward-only decoder with 512 hidden unit
    - 20M parameter

Experiment

- Train 2 speech recognition model
  - Model
    - MLF
  - Encode
    - 4 la
      -
    - 2 dc
  - Decode
    - 20M

The need for Bi-directional GRUs



- Bi-directional GRUs are just putting two independent GRUs together
- The input sequence is fed in forward order for one GRU, and in reverse order for the other
- The outputs of the two networks are usually concatenated at each time step
- Preserving information from both past and future helps understand context better

Ref(https://vtiya.medium.com/gru-vs-bi-gru-which-one-is-going-to-win-58a45ede5fba)

Experiment

- Train 2 speech recognition model
  - Model 2:
    - General attention model
  - Encoder:
    - 5 layer of uni-directional LSTM
    - 800 hidden units
    - 4 down-sampling pooling layer
  - Decoder: 2-layer forward-only decoder with 512 hidden unit
    - 30M parameter

Result

- WER(Word Error Rate)
  - The lower, the better

Table 1: Impact of quantizing the model's parameters into Posit and other popular data types on speech recognition performance during inference

Data type and Bitwidth	WER of Model 1	WER of Model 2
Native (IEEE754 FP32)	18.80	26.28
8-bit fixed-point<2,6>	22.33	27.25
8-bit fixed-point<3,5>	37.05	-
8-bit posit<8,0>	**19.10**	**26.28**
IEEE754 FP16	18.80	26.29
Bfloat16	18.97	26.36
16-bit fixed-point<2,14>	18.80	26.27
16-bit fixed-point<3,13>	18.78	-
16-bit posit<16,1>	**18.80**	**26.28**

Result

- Observation:
- posit<8,0> have small accuracy loss compare to original IEEE FP32
- For 8-bit data type, posit<8,0> outperform others.
- For all 16-bit data-type, the accuracy is about the same.
  - Expected since
    - weight is [-2.5, 2.5] for model 1
    - weight is [-2, 2] for model 2

Result

- Observation:
- 8-bit fixed-point<2,6> is not enough number > 2
  - But better than fixed-point<3,5>
  - Denser distribution in [-2, 2] in model 1
- After 1 epoch of re-train and re-quantize
  - model 1 accuracy: (19.10 -> 18.84) vs 18.80

Hardware Aspect

- Create hardware for:
  - Conversion between posit, floating and fixed point number
  - MAC (Multiply Accumulate)
    - a <- a+b*c
    - Good for convolution, matrix operation.

Hardware Spec

- 8-bit posit adder is approximately 25% of 32-bit float-point adder
- Posit adder is smaller than fixed-point adder

Table 3: Cost of float-to-posit and posit-to-float converter

Type	Power(nW)	Area(μm^2)
32-bit Float to 8-bit Posit	19841.83	774.49
8 bit Posit to 32-bit Float	19211.70	695.01

Table 4: Cost of posit, fixed-point, float-point adder and posit multiplier

Type	Power(nW)	Area(μm^2)
8-bit Posit Adder	7847.93	274.32
8 bit Posit Multiplier	298139.76	5090.91
8-bit Fixed-point Adder	8465.93	336.58
32-bit Float-point Adder	27753.62	1153.33

Hardware Spec

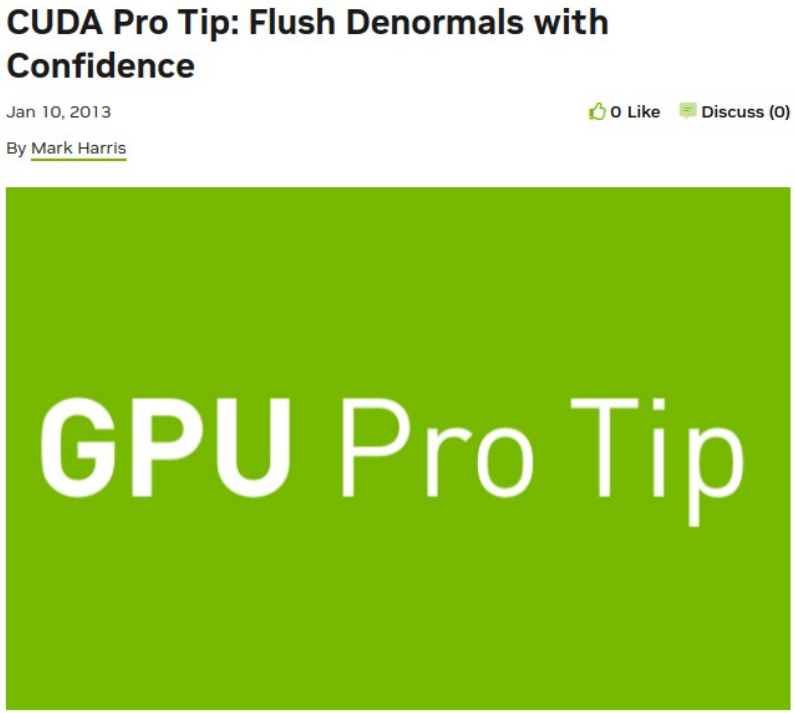
- 90nm library. 100MHz
- My 64-bit double multiplier: 5GHz, 111.2mW, 6.324um^2, 60 cycle. 16nm TSMC (No handling denormal)
- Original paper does not state that it's single cycle.

Table 4: Cost of posit, fixed-point, float-point adder and posit multiplier

Type	Power(nW)	Area(μm^2)
8-bit Posit Adder	7847.93	274.32
8 bit Posit Multiplier	298139.76	5090.91
8-bit Fixed-point Adder	8465.93	336.58
32-bit Float-point Adder	27753.62	1153.33

Hardware Spec

- Not handling denormal, +20% speedup



- ref:https://developer.nvidia.com/blog/cuda-pro-tip-flush-denormals-confidence/

Hardware Spec

- How many cycle of this hardware?
- What standard of the implementation did they follow?

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**

https://arxiv.org/abs/1811.01721

Prototype LSTM Engine Accelerator

- First 4 formula is similar to fully-connected layers, just matrix multiplication and addition. Can be parallelized
- Activation function is tanh and sigmoid, perform after summation of matrix multiplication
- Fifth formula is cell state, depend on previous state
- Last formula is for the next hidden state

x: input
w: weight
h: hidden state
c: cell state

$$i_t = \sigma\left(W_{ii}x_t + W_{hi}h_{t-1}\right) \quad (3)$$

$$f_t = \sigma\left(W_{if}x_t + W_{hf}h_{t-1}\right) \quad (4)$$

$$g_t = \tanh\left(W_{ig}x_t + W_{hg}h_{t-1}\right) \quad (5)$$

$$o_t = \sigma\left(W_{io}x_t + W_{ho}h_{t-1}\right) \quad (6)$$

$$c_t = \sigma\left(f_t * c_{t-1} + i_t * g_t\right) \quad (7)$$

$$h_t = \tanh(c_t) * o_t \quad (8)$$

Prototype LSTM Engine Accelerator

- Datapath:
  - Use cache buffers for input, hidden state, and weight. use 8MACs to calculate the matrix-multiplication and 4 accumulator for addition.
  - Feed 4 result to calculate activation.

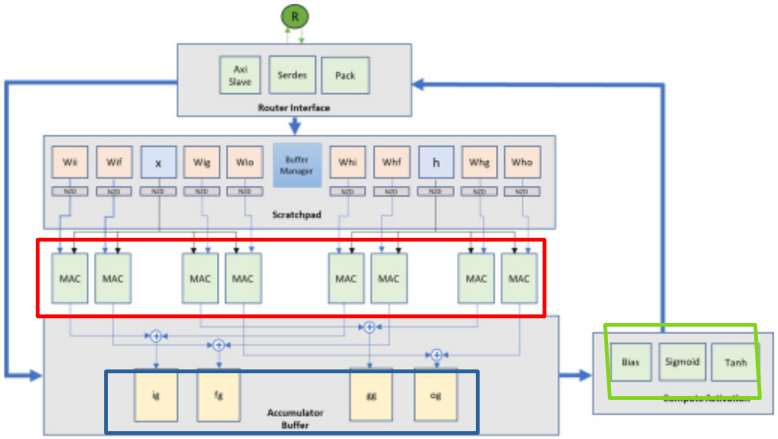


Figure 4: LSTM Accelerator Engine

$$i_t = \sigma\left(W_{ii}x_t + W_{hi}h_{t-1}\right) \quad (3)$$

$$f_t = \sigma\left(W_{if}x_t + W_{hf}h_{t-1}\right) \quad (4)$$

$$g_t = \tanh\left(W_{ig}x_t + W_{hg}h_{t-1}\right) \quad (5)$$

$$o_t = \sigma\left(W_{io}x_t + W_{ho}h_{t-1}\right) \quad (6)$$

$$c_t = \sigma\left(f_t * c_{t-1} + i_t * g_t\right) \quad (7)$$

$$h_t = \tanh(c_t) * o_t \quad (8)$$

Prototype LSTM Engine Accelerator

- Activation function tanh and sigma is done by piecewise approximation.
  - Using straight line segment to approximate the function.

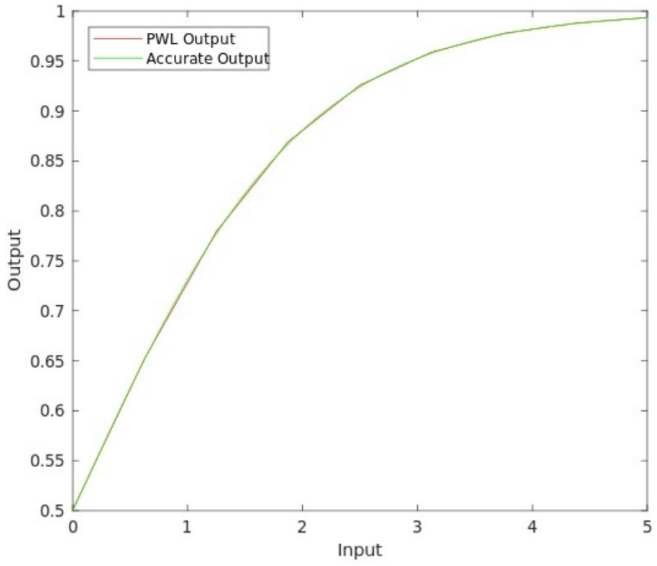


Figure 5: Sigmoid PWL Output vs. Accurate Sigmoid Output [2]

Summary of LSTM Engine Accelerator

- Synthesized the accelerator from ststemC code to RTL in verilog and verify the result.
- Bottleneck:
  - Storage of weight matrices.
  - Taking advantage of weight reuse across timesteps.

Conclusion:

- posit<8, 0> is suit for speech recognition inference.
- posit based hardware would have less area and power usage.
- posits present as appealing solution for compress DNN on edge and warrant further studies.

Question

- Does it work on other type of model?
  - ref: Rethinking floating point for deep learning
  - batch normalization fused into affine layers.
  - float32 parameters and network input are converted to our formats via round-to-nearest-even

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

Question

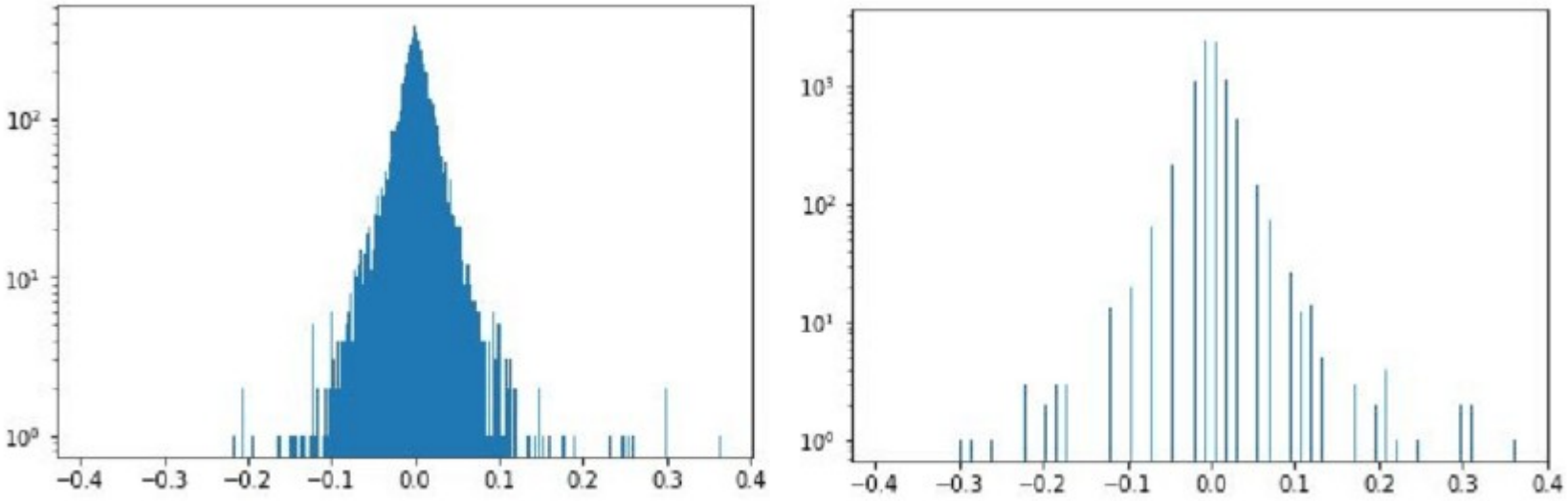
- Does it work on other type of model?
  - ref: Deep PeNSieve: a Deep learning framework based on the Posit Number System
  - Linear mapping quantization

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

	MNIST		Fashion-MNIST		SVHN		CIFAR-10	
Format	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$\langle 8,0\rangle$	98.77%	99.99%	88.52%	99.82%	81.31%	97.07%	43.89%	86.49%
Posit$\langle 8,0\rangle_{\text{quire}}$	99.07%	99.99%	89.92%	99.81%	89.13%	98.39%	68.88%	96.47%

Question

- Analyze different weight distribution on different operator and different model?
  - - Distribution of weights of ResNet-50 neural network before the quantization procedure (left) and after it (right).



https://www.researchgate.net/publication/329798596_Fast_Adjustable_Threshold_For_Uniform_Neural_Network_Quantization

Question

- Different quantize ways, like how it maps to uint8.
  - Dynamic/Static ways. (w/ and w/o callibration)
- Different hardware implementation can make the area and power have different outcome.

Thoughts

- You can make your whole model binary as long as your network has enough number of nodes:

Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to $+1$ or -1

Matthieu Courbariaux[*1] MATTHIEU.COURBARIAUX@GMAIL.COM
Itay Hubara[*2] ITAYHUBARA@GMAIL.COM
Daniel Soudry[3] DANIEL.SOUDRY@GMAIL.COM
Ran El-Yaniv[2] RANI@CS.TECHNION.AC.IL
Yoshua Bengio[1,4] YOSHUA.UMONTREAL@GMAIL.COM

[1]Université de Montréal
[2]Technion - Israel Institute of Technology
[3]Columbia University
[4]CIFAR Senior Fellow
*Indicates equal contribution. Ordering determined by coin flip.

Abstract

We introduce a method to train Binarized Neural Networks (BNNs) - neural networks with binary weights and activations at run-time. At training-time the binary weights and activations are used for computing the parameters gradients. During the forward pass, BNNs drastically reduce memory size and accesses, and replace most arithmetic operations with bit-wise operations, which is expected to substantially improve power-efficiency. To validate the effectiveness of BNNs we conduct two sets of experiments on the Torch7 and Theano frameworks. On both, BNNs achieved nearly state-of-the-art results over the

tistical machine translation (Devlin et al., 2014; Sutskever et al., 2014; Bahdanau et al., 2015), Atari and Go games (Mnih et al., 2015; Silver et al., 2016), and even abstract art (Mordvintsev et al., 2015).

Today, DNNs are almost exclusively trained on one or many very fast and power-hungry Graphic Processing Units (GPUs) (Coates et al., 2013). As a result, it is often a challenge to run DNNs on target low-power devices, and substantial research efforts are invested in speeding up DNNs at run-time on both general-purpose (Vanhoucke et al., 2011; Gong et al., 2014; Romero et al., 2014; Han et al., 2015) and specialized computer hardware (Farabet et al., 2011a;b; Pham et al., 2012; Chen et al., 2014a;b; Esser et al., 2015).

Thoughts

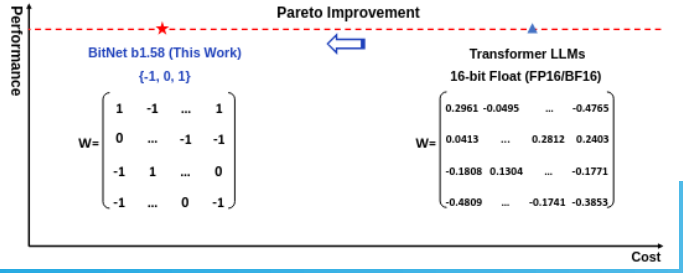
- You can 1-bit LLM, if you do quantization every linear layer and good at math:

The Era of 1-bit LLMs:
All Large Language Models are in 1.58 Bits

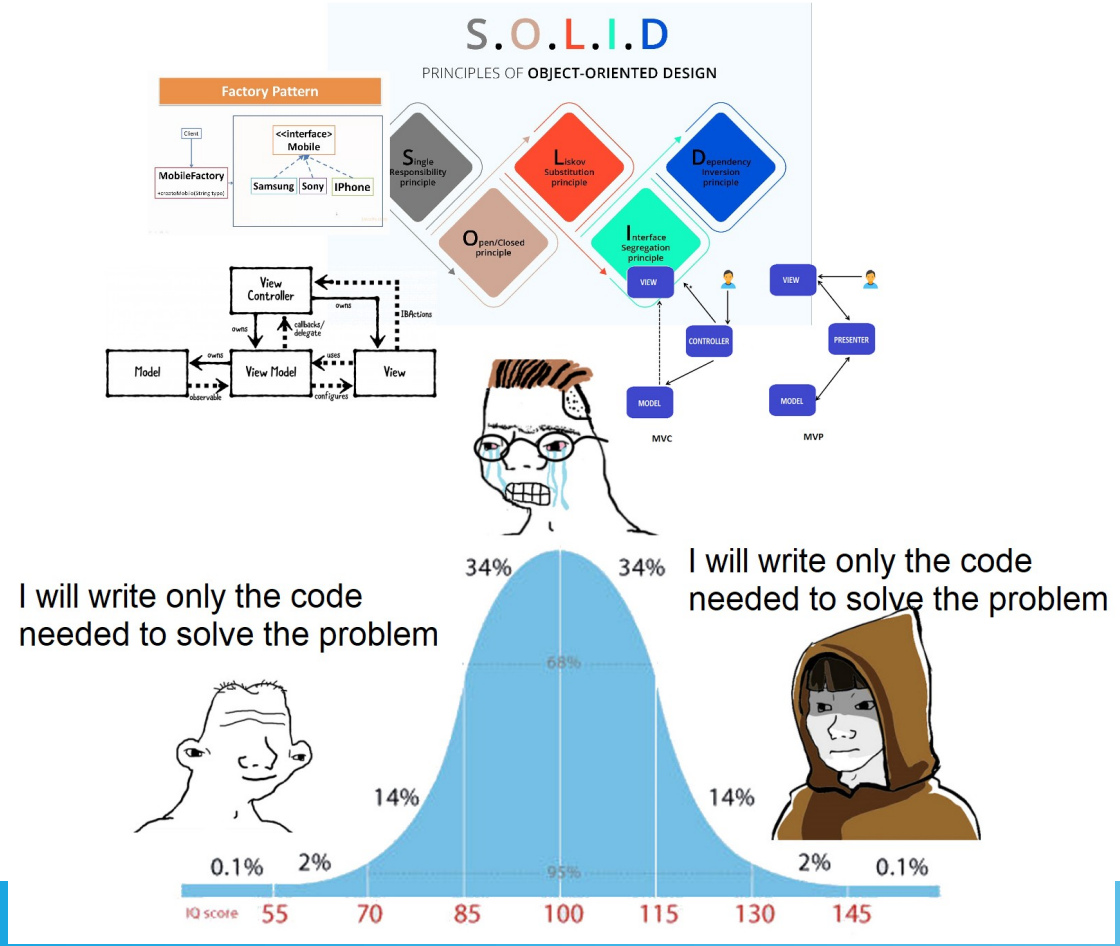
Shuming Ma* **Hongyu Wang*** **Lingxiao Ma** **Lei Wang** **Wenhui Wang**
Shaohan Huang **Li Dong** **Ruiping Wang** **Jilong Xue** **Furu Wei◇**
https://aka.ms/GeneralAI

Abstract

Recent research, such as BitNet [WMD+23], is paving the way for a new era of 1-bit Large Language Models (LLMs). In this work, we introduce a 1-bit LLM variant, namely **BitNet b1.58**, in which every single parameter (or weight) of the LLM is ternary {-1, 0, 1}. It matches the full-precision (i.e., FP16 or BF16) Transformer LLM with the same model size and training tokens in terms of both perplexity and end-task performance, while being significantly more cost-effective in terms of latency, memory, throughput, and energy consumption. More profoundly, the 1.58-bit LLM defines a new scaling law and recipe for training new generations of LLMs that are both high-performance and cost-effective. Furthermore, it enables a new computation paradigm and opens the door for designing specific hardware optimized for 1-bit LLMs.



Thank You


S.O.L.I.D
PRINCIPLES OF OBJECT-ORIENTED DESIGN
Factory Pattern
Client
MobileFactory
<<interface>>
Mobile
Samsung
Sony
IPhone
Single Responsibility principle
Open/Closed principle
Liskov Substitution principle
Interface Segregation principle
Dependency inversion principle
View Controller
callbacks/ delegate
IBActions
Model
View Model
View
VIEW
CONTROLLER
MODEL
MVC
PRESENTER
MVP
I will write only the code needed to solve the problem
34%
34%
68%
14%
14%
0.1%
2%
95%
2%
0.1%
IQ score
55
70
85
100
115
130
145
I will write only the code needed to solve the problem