Adaptive Posit: Parameter aware numerical format for deep learning inference on the edge

Hamed F. Langroudi [§,†], Vedant Karia [§], John L. Gustafson[*], Dhireesha Kudithipudi[§]
[§] Neuromorphic AI Lab, University of Texas at San Antonio, TX, USA
[†] Rochester Institute of Technology, NY, USA
[*]National University of Singapore, Singapore

Abstract

Ultra low-precision ($<$ 8-bit width) arithmetic is a discernible approach to deploy deep learning networks on to edge devices. Recent findings show that posit with linear quantization has similar dynamic range as the weight and activation values across the deep neural network layers. This characteristic can benefit the data representation of deep neural networks without impacting the overall accuracy. When capturing the full dynamic range of weights and activations, posit with mixed precision or linear quantization leads to a surge in hardware resource requirements.

We propose adaptive posit, which has the ability to capture the non-homogeneous dynamic range of weights and activation's across the deep neural network layers. A fine granular control is achieved by embedding the hyperparameters in the numerical format. To evaluate the overall system efficiency, we design a parameterized ASIC soft core for the adaptive posit encoder and decoder. Benchmarking and evaluation of the adaptive posit is performed on three datasets: Fashion-MNIST, CIFAR-10, and ImageNet. Results assert that on average the performance on inference with $<$ 8-bit adaptive posits surpasses (2% to 10%) that of posit.

1. Introduction

Deep neural networks (DNN) are used in a wide range of applications such as recommender systems [28] to precision agriculture [23]. The fast prediction during inference demands intensive memory and compute resources on high-end compute platforms. Conventional approaches of deploying the computations to the cloud raise concerns over privacy, latency, and energy efficiency [25]. To amortize these costs, several lightweight DNN models have been proposed recently [6, 14, 16, 24].

Among these models, low-precision DNN is one of the most promising approaches to address the energy constraints of edge devices, where it compacts the deep learning parameters and speeds up computation [16]. However, the

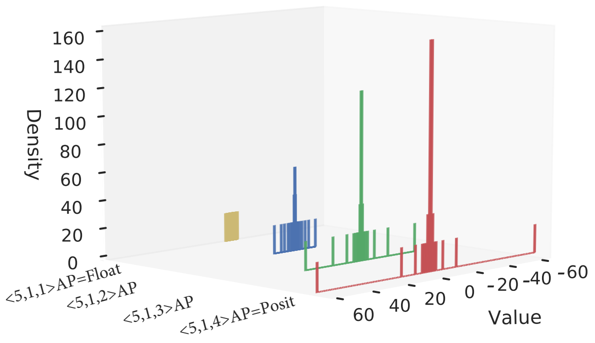

Figure 1: An illustration of the average and standard deviation for $< n, es, rs >$ adaptive posit (AP) representation, where $n = 5$ bits, $es = 1$ exponent bits and $rs = [1, 2]$ bits. Simulations performed on ResNet18 with ImageNet.

performance of ultra-low-precision ($<$ 8-bit width) models is degraded significantly when hardware-oriented numerical formats such as binary/ternary, fixed-point, etc. are utilized to represent DNN parameters [22]. Dissimilarity between DNN parameters and the low precision fixed-point numerical format representations, in terms of distribution (uniform or non-uniform), dynamic range, and precision are the primary contributing factors to this degradation in accuracy [22]. An alternate solution is to use other numerical formats whose values have similar statistics as DNN parameters. Posit [11] is one of these numerical formats which has recently shown potential benefits for $[6..8]$-bit width data representation and computation of DNNs without impacting the overall accuracy [19]. However, performing DNN inference with posit using a 5-bit representation of weights, also fails to preserve accuracy when the complexity of benchmarks and neural networks is increased [19]. For instance, the performance of DNN inference on the CIFAR-10 dataset with an 8-layer DNN drops from 92.10% to 15.18% while using 5-bit weights in the posit format as compared to the high-precision 32-bit floating point in similar benchmark. One plausible reason for the performance drop could be the inability to capture the variability in the dynamic range of DNN parameters across the layers.

Approaches to mitigate this problem, such as linear, or mixed-precision quantization, [16] and numerical format scaling [20, 26] increase the computational complexity. One approach to ameliorate this problem is to ensure that dynamic range of the numerical format inherently matches the statistics of the DNN parameters. The adaptive posit numerical format proposed in this research offers such flexibility, wherein the dynamic range can be tuned (Fig. 1). In this introductory paper, we are motivated to evaluate the efficacy of the adaptive posit representation as compared to the standard posit numerical format to represent weights and activations in various DNN models for image classification. Note the following three reasons which motivated us to evaluate adaptive posit for representation purposes in this research while we explore adaptive posit for computation in the future work: (i) The commercial hardware architecture used in edge devices mostly support floating point or fixed-point. Posit numerical format is not offered yet because of its recent introduction [5]; (ii) The total energy consumption of the DNN models on edge devices is dominated by memory operations [27]. For instance, in a 28 nm CMOS process, the 16-bit SRAM (32 K memory size) memory access requires 80x more energy than the MAC operations. A 16-bit DRAM access requires 2.6 orders of magnitude more energy than the MAC operations [27]; and (iii) Edge devices support a small memory footprint (less than 1 MB).

This research article highlights the following contributions:

- We propose a new numerical format, adaptive posit, that is capable of adapting its dynamic range to match that of the weights and activations in each DNN layer.
- We update the MemPosit architecture [21] to support two new conversion algorithms; adaptive posit to high-precision floating point and vice versa.
- We show that [5..8]-bit adaptive posit is efficient over posit when benchmarked for latency and energy across three data corpus.

2. Related Work

Since as early as the 1980s, low-precision arithmetic has been explored as an option for reducing memory and compute complexity in shallow neural networks without having to sacrifice performance [2, 15]. In some scenarios, it has even been shown to improve performance, as the quantization noise generated from the use of low-precision parameters in shallow neural networks acts as a regularization method [2]. The outcome of these studies indicates that 16 and 8-bit precision DNN parameters are sufficient for training and inference respectively, on shallow networks [2, 15]. The capability of low-precision arithmetic is being reevaluated in the deep learning era to verify whether the earlier mentioned benefits are still able viable for reducing the memory footprint and energy consumption during training and inference [3, 4, 7, 8, 10, 12, 13, 17–19, 21].

The performance of DNN inference without retraining is more robust to the noise that is generated from low-precision DNN parameters as they are static during inference; several groups have demonstrated that either 8-bit block floating point (BFP) or 8-bit fixed-point, coupled with linear quantization, are adequate to represent weights and activations without significantly degrading the performance yielded with 32-bit floating point. Note that the accumulation bit-width is selected to be 32 to preserve accuracy while performing thousands of general purpose MAC-based addition operations. For instance, Gysel *et al.* demonstrates that an 8-bit BFP for representing weights and activations, in 8-bit multipliers, and 32-bit accumulators results in $<1\%$ loss in accuracy on AlexNet with the ImageNet corpus [12]. Following this work, Hashemi *et al.* introduced low-precision DNN inference networks to better understand the impact of numerical formats on the energy consumption and performance of DNNs [12, 13]. For instance, performing inference on AlexNet with the 32-bit fixed-point format yields a $6\times$ more energy consumption over 8-bit fixed-point for the CIFAR-10 dataset [13]. Chung *et al.* proposed the Brainwave accelerator using 8-bit BFP with a 5-bit exponent to classify ImageNet dataset on ResNet-50 with $<2\%$ accuracy loss [7]. However, the scaling factor parameter in the BFP numerical format needs to be updated according to the DNN parameter statistics, thus increasing the computational complexity of inference.

To alleviate this problem, researchers have used posits in DNNs [3, 4, 17–19, 21]. Posits represent numbers more accurately around ± 1 and less accurately for very small and large numbers, unlike the uniform precision of the floating point [11]. This tapered precision characteristic of posits suits the distribution of DNN parameters well [11, 21]. For instance, Langroudi *et al.* explored the efficacy of posits to represent DNN weights and have shown that it is possible to achieve performance within $<1\%$ variation on the AlexNet and ImageNet corpora with 7-bit weight representation [21]. They also demonstrate that posits have 30% lower memory footprint than fixed-point across multiple DNNs while maintaining $<1\%$ drop in accuracy. However, this work uses 7-bit posit quantized weights that are converted to 32-bit floats, restricting posits for memory storage only.

To take full advantage of the posit numerical format, Carmichael *et al.* proposed the Deep Positron DNN accelerator which employs the posit numerical format to represent weights and activations combined with an FPGA-based soft-core for $\leq$8-bit precision exact-MAC operations [3, 4]. They demonstrate that 8-bit posits outperform both 8-bit

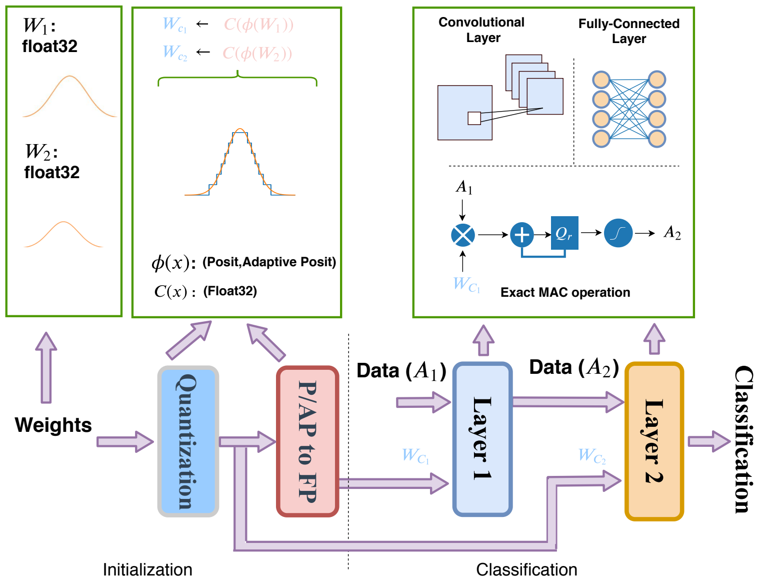

Figure 2: The *Modified MemPosit* framework for DNN inference with two layers. The framework scales to any DNN architecture. P= Posit; AP= Adaptive Posit; FP= Floating Point.

fixed-point and floating point numbers on low-dimensional datasets, such as Iris [9]. The later study is extended to ultra-low precision ([5..8]-bit) and high-dimensional datasets such as MNIST, Fashion-MNIST, and CIFAR-10 by Langroudi *et al.*, which consistently shows the advantages of posits over other numerical formats at ultra-low precision [18]. Following these works, Jeff Johnson proposed a log float format that combines the posit numerical format with exact log-linear multiply-add (ELMA), which is the logarithmic version of the exact MAC operation. This work shows that it is indeed possible to classify ImageNet with the ResNet DNN architecture with $<1\%$ degradation in accuracy [17]. Finally, the low-precision posit arithmetic is extended to both DNN training and inference with different quantization approaches for both feedforward and convolution neural networks in Cheetah framework, and is evaluated on various datasets [19].

However, posit is unable to capture the variance in dynamic range of a DNN's parameters. Therefore, the proposed research explores the efficacy of the adaptive posit numerical format for DNNs with low-precision representation of weights and activations in variety of image classification.

3. Adaptive Posit Numerical Format

The posit, a Type III unum, is a new numerical format with a tapered precision characteristic which was proposed as an alternative to IEEE-754 floating format to represent real numbers [11]. Posit revamped the IEEE-754 floating format and addressed complaints about Type I and Type II unums. Posits provides better accuracy, dynamic range, and program reproducibility than IEEE floating point. The essential advantage of posits is their capability to represent non-linearly distributed numbers in a specific dynamic range around 1 with maximum accuracy. The value of a posit number is represented by Equation (1), where s represents the sign, es and fs represent the maximum number of bits allocated for the exponent and fraction, respectively, e and f indicate the exponent and fraction values, respectively, and k, as computed by Equation (2), represents the regime value.

$$x = \begin{cases} 0, & \text{if } (\texttt{00...0}) \\ NaR, & \text{if } (\texttt{10...0}) \\ (-1)^s \times 2^{2^{es} \times k} \times 2^e \times \left(1 + \frac{f}{2^{fs}}\right), & \text{otherwise} \end{cases} \quad (1)$$

The regime bit-field is encoded based on the *runlength* m of identical bits $(r...r)$ terminated by either a *regime terminating bit* $\overline{r}$ or the end of the n-bit value. Note that there is no requirement to distinguish between negative and positive zero since only a single bit pattern $(\texttt{00...0})$ represents zero. Furthermore, instead of defining a *NaN* for exceptional values and infinity by different bit patterns, a single bit pattern $(\texttt{10...0})$, "Not-a-Real" (NaR), represents exception values and infinity. More details about the posit number format can be found in [11].

$$k = \begin{cases} -m, & \text{if } r = 0 \\ m-1, & \text{if } r = 1 \end{cases} \quad (2)$$

The adaptive posit numerical format is a version of the posit format where the dynamic range is controlled by a new hyperparameter. This hyperparameter can be defined either

Algorithm 1 Adaptive posit encoder for converting a real value to n-bit outputs with es exponent bits and $\log n$ rs bits.
Input: real value
Output: <n,es,rs> adaptive posit numerical format

```
 1: procedure ENCODE(in)                      ▷ Data extraction of input
 2:     nzero ← |in                           ▷ '1' if in is nonzero
 3:     sign ← in > 0 ? 0 : 1
 4:     Abs ← Abs(in)
        Encode the regime bit
 5:     if $in > 1$ then
 6:         RunLength ← 1
 7:         while $in > 2^{2^{es}}$ & $RunLength > rs$ do
 8:             in← in/$2^{2^{es}}$
 9:             RunLength ← RunLength + 1
10:         end while
11:         reg ← RunLength − 1                ▷ set Regime bit
12:     else
13:         RunLength ← 0
14:         while $in < 1$ & $RunLength > rs$ do
15:             in← in×$2^{2^{es}}$
16:             RunLength ← RunLength + 1
17:         end while
18:         reg ← −1 × RunLength              ▷ set Regime bit
19:     end if
        Encode the exponent bit
20:     e ← $2^{es-1}$
21:     while $e > 0.5$ do
22:         if $in < 2^e$ then
23:             in ← in/$2^e$
24:             exp ← exp + 1
25:         end if
26:         e ← e/2
27:     end while
        Encode the fraction
28:     frac ← RNE(in − 1)                    ▷ Round-Tie-Even
29:     return sign, reg, exp, frac
30: end procedure
```

as a regime bias parameter called K_b (in a range of $[0, n-2]$), similar to the normalized posit format [20] or the adaptive float numerical format [26], or as a maximum regime bit-width called rs (in a range of $[1, n-1]$). Depending on which hyperparameter is used, the maximum and minimum positive value that is represented by this numerical format is computed either by Equations (3) and (4) or by Equations (5) and (6) where $t = n - rs - 1$. Fundamentally, adaptive posit can represent float numerical format ($rs = 1$), posit ($rs = n - 1$) and other tapered numerical precision format between them.

Algorithm 2 Adaptive posit decoder for converting an n-bit input with es exponent bits, $\log n$ rs bits into a real value.
Input: $< n, es, rs >$ adaptive posit numerical format
Output: real value

```
 1: procedure DECODE(in)                      ▷ Data extraction of input
 2:     nzero ← |in                           ▷ '1' if in is nonzero
 3:     sign ← in[$n$−1]                      ▷ Extract sign
 4:     twos ← ({$n$−1{sign}} ⊕ in[$n$−2 : 0]) + sign
 5:     rc ← twos[$n$−2]                      ▷ Regime check
 6:     inv ← {$n$−1{rc}} ⊕ twos              ▷ Invert 2's
        Extract regime value with $rs$
 7:     zc ← LZD(inv[n−1 : n−1 − rs])
 8:     reg ← rc ? zc−1 : −zc
        Extract exponent and fraction
 9:     tmp ← twos[$n$−2 : 0] ≪ (zc + 1)
10:     exp ← tmp[$n$−2 : $n$−$es$−1]
11:     frac ← {nzero, tmp[$n$−$es$−2 : 0]}
12:     return $(-1)^{\text{sign}} \times 2^{2^{\text{es}} \times \text{reg}} \times 2^{\text{exp}} \times \left(1 + \frac{\text{frac}}{2^{\text{fs}}}\right)$
13: end procedure
```

$$Max(x_{AP}) = 2^{2^{es} \times (k - K_b)} \tag{3}$$

$$Min(x_{AP}) = 2^{-2^{es} \times (k - K_b)} \tag{4}$$

$$Max(x_{AP}) = \begin{cases} 2^{2^{es} \times (rs-1)}, & \text{if } (t = 0) \\ 2^{2^{es} \times (rs - 2^{-t})}, & \text{if } (t \le es) \\ 2^{2^{es} \times rs} \times (1 - 2^{es-t-1}) & \text{otherwise} \end{cases} \tag{5}$$

$$Min(x_{AP}) = \begin{cases} 2^{-2^{es} \times (rs-1)}, & \text{if } (t = 0) \\ 2^{-2^{es} \times (rs - 2^{-t})} & \text{if } (t \le es) \\ 2^{-2^{es} \times rs} \times (1 + 2^{es-t}), & \text{otherwise} \end{cases} \tag{6}$$

4. DNN with Adaptive Posit weight representation

The MemPosit framework [20] is modified (as shown in Fig. 2) to achieve adaptive posit weight and activation representation in performing DNN inference. The MemPosit is divided into two sub-modules (initialization and classification). In the initialization step, the 32-bit precision floating point learned weights are quantized to low-precision adaptive posit numerical format (Algorithm 1). In this paper, the quantization function is defined by Equation (7) where δ and λ are the maximum and minimum values represented in low-precision numerical format. To perform image classification by DNN, the quantized weights and quantized activations that are computed in each layer are converted back to the 32-bit precision floating point (Algorithm 2). For brevity, the encoding and decoding algorithm explained here are based on two hidden layers DNN inference using the adaptive posit numerical format with the rs hyperparameter. When the K_b hyperparameter is used, the encoding and decoding are similar to posit with the exception of shifting regime value with K_b. Note that we do not consider "Not a Real" because all DNN parameters and data are real-valued, and adaptive posits do not overflow to infinity.

$$Q(W_i) = \begin{cases} \delta, & x > \delta \\ Float\_to\_Adaptive\ Posit & \delta > x > \lambda \\ \lambda, & \lambda > x \end{cases} \quad (7)$$

To encode adaptive posit, at the first step, the zero value and sign are determined. By capturing the sign bit, the absolute value of the real number is enough to determine other posit encode bits (lines 2-4). When the rs hyperparameter is selected, The regime bit (reg) is computed by $\leq rs$ times dividing or multiplying a real number by $2^{2^{es}}$ until the number is in the range [1, $2^{2^{es}}$) (lines 5-19). Otherwise, the aforementioned range condition is enough to terminate this process (lines 11-14). To find the exponent, this process is continued until the number is in the range [1, 2) (lines 20-27). To compute the fraction, the remaining value is diminished by one and rounded to the nearest even number.

To decode adaptive posit, during the initialization step (lines 2-6), the zero value and sign are captured, and the inverse of the two's complement of the input is calculated. The last step is performed to bypass the requirement to compute both Leading Zero Detection (LZD) and Leading One Detection (LOD). The regime value is extracted (lines 7-8). The regime bits are shifted out, and the exponent based on the es value and fraction is obtained (lines 9-12). Note that a *regime-terminating bit* $\overline{r}$ and extended zero are assumed if we run out of space (n bit-width) to compute the regime-bit as well as the exponent and the fraction bits.

5. ASIC Soft Core for Adaptive Posit

The encoding and decoding architecture, detailed in Figs 3, and 4, compute the conversion between adaptive posit and floating point numerical format and vice versa. The encoder and decoder design follows the algorithm described in Algorithms 1 and 2. Posit numbers are represented as a combination of sign, *rs*, *e* and fraction bits respectively. *rs* value is the number of leading zeros in the result obtained by performing XOR operation on every consecutive pair of bits in the input. The input is left shifted *rs* times to extract the *es*. The exponent value is computed as a product of *rs* and 2^{es} summed with *e*. The multiplication is performed by left shifting the *rs* bits by *es*. The remaining bits post the calculation of the exponential bits are assigned to fraction bits of float value. To encode adaptive posit, the number of XOR operations are reduced from $n - 1$ to rs during the extraction of regime bits. However, rest of the process remains same as posit encoding. The hardware overhead increases because of the circuitry needed for selecting arbitrary rs.

For encoding a 32-bit float value, the exponent value of float is subtracted by 127 to extract *rs* and *e* values. The most significant bit of the subtraction operation indicates the sign of the *rs* value. Based on the sign of *rs*, either 01 or 10 is concatenated to the fraction and *e* bits respectively, and arithmetic right shift is performed on the concatenated value. To increase the precision of the decoding, the fraction bits are rounded by introducing the sticky bit. Sticky bit is calculated by performing OR operation on the fraction bits that were not accommodated in the encoded value. Adding the sticky bit to the encoded value rounds off to the closest value. The rounding off adds an overhead on the power and the delay of the hardware.

6. Simulation Results & Analysis

A modified version of the MemPosit [21] is implemented in Python and extended to the Tensorflow framework [1]. The performance and efficiency of adaptive posit and posit numerical formats is evaluated for three DNN inference tasks. The specification of the tasks and inference performance on 32-bit floating point DNNs are summarized in Table 1. Table 2 presents the low-precision inference performance for the three data corpus on posit and adaptive posit formats.

The findings show that the ultra low-precision adaptive posit (with es=1) outperforms the posit on most tasks and DNN models. For instance, the 5-bit low-precision performance of ResNet-50 on CIFAR-10 dataset is boosted by 36.31% for adaptive posit weight representation in comparison to posit. This boost up, however is not observed in high-dimensional datasets such as ImageNet. The weights skewed towards zero may be a plausible cause for this observation. A combination of adaptive posit and linear quantization may address this drawback.

To compare the footprint of these two numerical formats, the maximum frequency and power of the encoder and the decoder designs of these numeric formats is measured using Synopsys design compiler as shown in Fig. 5. The results for adaptive posit show a 5% improvement over posit in the decoder's maximum frequency while consuming the same power. This improvement is attributed to the reduction in the number of XOR operations for computing the regime bit. The encoder design, however does not show any difference in performance or efficiency over posit format.

The results show that adaptive posit is able to achieve a 10% boost up in performance over posit numerical format. Additionally, systems performing inference by using adaptive posit are able to exhibit efficient processing over posit when benchmarked for latency and power.

In summary, the best trade-off between maximum fre-

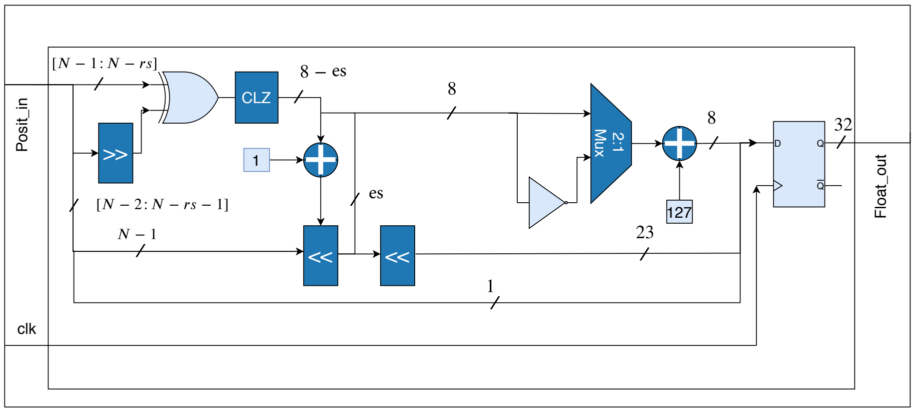

Figure 3: RTL design for converting N-bit adaptive posit with es exponential bits and rs regime bits to floating point single precision format. Floating point exponent value is combination of rs and es bits which is computed by counting the number of leading zeros. Adaptive posit fraction bits are directly assigned floating point fraction bits.

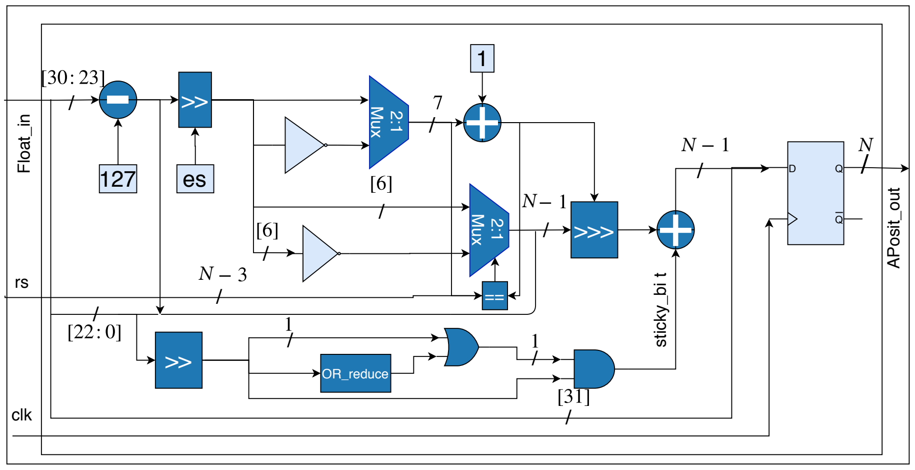

Figure 4: RTL design for converting floating point single precision to adaptive posit format with rounding off fraction bits using sticky bit. Regime bits are computed by performing arithmetic right shift.

Table 1: Specifications of the benchmark tasks and performance on a baseline 32-bit floating point network

Dataset	Layers[1]	# Parameters	# EMAC Ops[2]	Memory	Accuracy
Fashion-MNIST	2 Conv, 3 FC, 2 PL, 1 BN	1.88 M	69.8 K	7.77 MB	92.54%
CIFAR-10	7 Conv, 1 FC, 3 PL	0.95 M	312.60 K	6.23 MB	81.37%
	ResNet-18	0.27 M	286.72 K	3.01 MB	91.54%
	ResNet-50	0.86 M	802.89 K	8.76 MB	92.10%
ImageNet	ResNet-18	11.70 M	3.43 M	72.69 MB	68.10%

[1] Conv: 2D convolutional layer; FC: fully-connected layer; PL: max/avg. pooling layer; BN: batch normalization layer.
[2] The number of EMAC operations for a single sample.

quency and average accuracy degradation from 32-bit floating point on all the benchmarks (when analyzed across the [5..8]-bit range) is achieved by utilizing 5-bit adaptive posit. Looking at the adaptive posit numerical format in terms of

Table 2: *MemPosit* accuracy on three datasets with [5..8]-bit precision adaptive posit weights compared to posit respective best results are when adaptive posit and posit have $es \in \{0, 1\}$

Dataset	DNN	Adaptive posit with rs				Posit				Adaptive posit with K_b			
		8-bit	7-bit	6-bit	5-bit	8-bit	7-bit	6-bit	5-bit	8-bit	7-bit	6-bit	5-bit
F-MNIST	Conv	92.61%	92.48%	92.48%	91.94%	92.60%	92.48%	**92.53%**	90.48%	**92.65%**	**92.55%**	92.51%	**92.25%**
	Conv	81.22%	80.17%	78.92%	71.28%	81.19%	80.16%	76.00%	71.30%	**81.47%**	**81.41%**	**79.97%**	**73.29%**
CIFAR-10	ResNet-18	**91.10%**	**90.68%**	**83.46%**	46.70 %	91.10%	90.64%	82.87%	48.57%	91.10%	90.64%	82.87%	**57.44%**
	ResNet-50	**91.65%**	**90.42%**	**75.95%**	**51.49%**	91.57%	90.17%	72.56%	15.18%	91.57%	90.17%	72.56%	15.18%
ImageNet	ResNet-18	**62.32%**	**12.65%**	**0.10%**	**0.10%**	62.32%	12.65%	0.10%	0.10%	62.32%	12.65%	0.10%	0.10 %

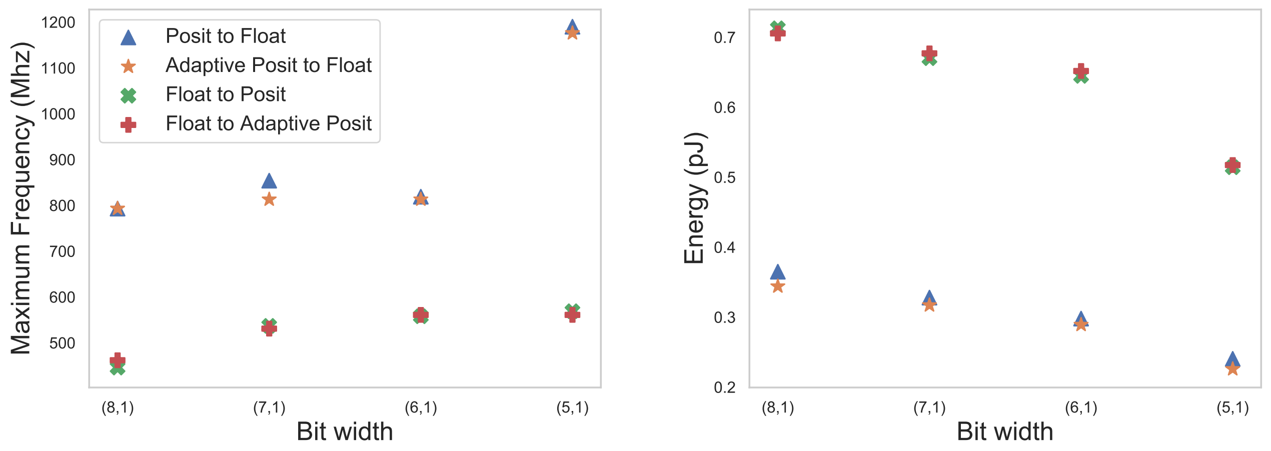

Figure 5: Analysis of encoding and decoding operations for adaptive posit with 4 rs bits and posit. (a) Maximum operating clock frequency of the different encoding blocks; (b) Energy requirements of the encoding and decoding architecture blocks.

classification performance and unit energy-delay-product, posits with $es = 1$ provide a better trade-off compared to posits with $es \in \{0, 2\}$ same as posit.

7. Conclusions

In this work, adaptive posit numerical format is studied in the context of DNNs for the edge. The adaptive posit format DNNs are realized on a modified version of *MemPosit* framework, where the network maintains a high degree of integrity at even [5-6] bit precision. The area and resource overhead introduced by the adaptive posit are negligible. This motivates us to further investigate ultra-low precision DNN training with adaptive posit.

8. Acknowledgements

The authors would like to thank Anurag Daram and Tej Pandit from Neuromophic AI lab for their support in this work.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[2] Krste Asanovic and Nelson Morgan. Experimental determination of precision requirements for back-propagation training of artificial neural networks, international computer science institute, 1991.

[3] Zachariah Carmichael, Hamed Fatemi Langroudi, Char Khazanov, Jeffrey Lillie, et al. Deep positron: A deep neural network using the posit number system. In *Design, Automation & Test in Europe Conference & Exhibition, DATE*, pages 1421–1426, Florence, Italy, Mar. 2019. IEEE.

[4] Zachariah Carmichael, Hamed F. Langroudi, Char Khazanov, Jeffrey Lillie, et al. Performance-efficiency trade-off of low-precision numerical formats in deep neural networks. In *Proceedings of the Conference for Next Generation Arithmetic*, CoNGA'19, pages 3:1–3:9, Singapore, Singapore, 2019. ACM.

[5] Rohit Chaurasiya, John Gustafson, Rahul Shrestha, Jonathan Neudorfer, et al. Parameterized posit arithmetic hardware generator. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 334–341. IEEE, 2018.

[6] Yunpeng Chen, Haoqi Fang, Bing Xu, Zhicheng Yan, et al. Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution. *arXiv preprint arXiv:1904.05049*, 2019.

[7] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, et al. Serving DNNs in real time at datacenter scale with Project Brainwave. *IEEE Micro*, 38(2):8–20, 2018.

[8] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Low precision arithmetic for deep learning. In *Workshop Track Proceedings of the 3rd International Conference on Learning Representations, ICLR*, San Diego, CA, USA, May 2015.

[9] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[10] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1737–1746, Lille, France, July 2015. JMLR.org.

[11] John L Gustafson and Isaac T Yonemoto. Beating floating point at its own game: Posit arithmetic. *Supercomputing Frontiers and Innovations*, 4(2):71–86, 2017.

[12] Philipp Gysel, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

[13] Soheil Hashemi, Nicholas Anthony, Hokchhay Tann, R. Iris Bahar, et al. Understanding the impact of precision quantization on the accuracy and energy of neural networks. In *Design, Automation & Test in Europe Conference & Exhibition, DATE*, pages 1474–1479, Lausanne, Switzerland, Mar. 2017. IEEE.

[14] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[15] Akira Iwata, Yukio Yoshida, Satoshi Matsuda, Yukimasa Sato, et al. An artificial neural network accelerator using general purpose 24 bits floating point digital signal processors. In *International Joint Conference on Neural Networks, IJCNN*, volume 2, pages 171–175, 1989.

[16] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[17] Jeff Johnson. Rethinking floating point for deep learning. *arXiv preprint arXiv:1811.01721*, 2018.

[18] H. F. Langroudi, Z. Carmichael, J. L. Gustafson, and D. Kudithipudi. Positnn framework: Tapered precision deep learning inference for the edge. In *2019 IEEE Space Computing Conference (SCC)*, pages 53–59, July 2019.

[19] Hamed F Langroudi, Zachariah Carmichael, David Pastuch, and Dhireesha Kudithipudi. Cheetah: Mixed low-precision hardware & software co-design framework for dnns on the edge. *arXiv preprint arXiv:1908.02386*, 2019.

[20] Seyed HF Langroudi, Tej Pandit, and Dhireesha Kudithipudi. Deep learning inference on embedded devices: Fixed-point vs posit. *arXiv preprint arXiv:1805.08624*, 2018.

[21] S. H. Fatemi Langroudi, T. Pandit, and D. Kudithipudi. Deep learning inference on embedded devices: Fixed-point vs posit. In *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, pages 19–23, March 2018.

[22] Asit Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*, 2017.

[23] Alex Olsen, Dmitry A Konovalov, Bronson Philippa, Peter Ridd, Jake C Wood, Jamie Johns, Wesley Banks, Benjamin Girgenti, Owen Kenny, James Whinney, et al. Deepweeds: A multiclass weed species image dataset for deep learning. *Scientific Reports*, 9(1):2058, 2019.

[24] Mengye Ren, Andrei Pokrovsky, Bin Yang, and Raquel Urtasun. Sbnet: Sparse blocks network for fast inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8711–8720, 2018.

[25] Shadab Siddiqui, Manuj Darbari, and Diwakar Yagyasen. A comprehensive study of challenges and issues in cloud computing. In *Soft Computing and Signal Processing*, pages 325–344. Springer, 2019.

[26] Thierry Tambe, En-Yu Yang, Zishen Wan, Yuntian Deng, Vijay Janapa Reddi, Alexander Rush, David

Brooks, and Gu-Yeon Wei. Adaptivfloat: A floating-point based data type for resilient deep learning inference. *arXiv preprint arXiv:1909.13271*, 2019.

[27] Xuan Yang, Mingyu Gao, Jing Pu, Ankita Nayak, Qiaoyi Liu, Steven Emberton Bell, Jeff Ou Setter, Kaidi Cao, Heonjae Ha, Christos Kozyrakis, et al. Dnn dataflow choice is overrated. *arXiv preprint arXiv:1809.04070*, 2018.

[28] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.